The NASA STI Program Office … in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

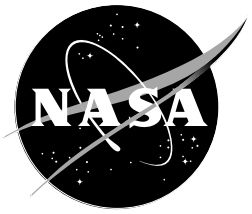
- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and mission, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at http://www.sti.nasa.gov/STI-homepage.html
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at (301) 621-0134
- Telephone the NASA Access Help Desk at (301) 621-0390
- Write to:
  NASA Access Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/CP—2004–212750



NASA/IEEE MSST 2004
Twelfth NASA Goddard Conference on Mass Storage Systems and Technologies

in cooperation with the

Twenty-First IEEE Conference on Mass Storage Systems and Technologies

Edited by
Ben Kobler, Goddard Space Flight Center, Greenbelt, Maryland
P C Hariharan, Systems Engineering and Security, Inc., Greenbelt, Maryland

Proceedings of a conference held at
The Inn and Conference Center
University of Maryland, University College,
Adelphi, Maryland, USA
April 13-16, 2004

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

April 2004

Available from:

NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320
Price Code: A17

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Price Code: A10

Preface

MSST2004, the Twelfth NASA Goddard / Twenty-first IEEE Conference on Mass Storage Systems and Technologies has as its focus long-term stewardship of globally-distributed storage. The increasing prevalence of *e-anything* brought about by widespread use of applications based, among others, on the World Wide Web, has contributed to rapid growth of online data holdings. A study[1] released by the School of Information Management and Systems at the University of California, Berkeley, estimates that over 5 exabytes of data was created in 2002. Almost 99 percent of this information originally appeared on magnetic media. The theme for MSST2004 is therefore both timely and appropriate. There have been many discussions about rapid technological obsolescence, incompatible formats and inadequate attention to the permanent preservation of knowledge committed to digital storage. Tutorial sessions at MSST2004 detail some of these concerns, and steps being taken to alleviate them. Over 30 papers deal with topics as diverse as performance, file systems, and stewardship and preservation. A number of short papers, extemporaneous presentations, and works in progress will detail current and relevant research on the MSST2004 theme.

Our thanks go to the researchers, authors, and the Program Committee for their zeal and energy in putting together an interesting agenda.

P C Hariharan
Nabil Adam
Publications Chairs

Ben Kobler
Conference Chair

1 http://www.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm

Organizing Committee

Conference and Program Committee Chair
Ben Kobler, NASA Goddard Space Flight Center

Program Committee
Ahmed Amer, *University of Pittsburgh*
Curtis Anderson, *Mendocino Software*
Jean-Jacques Bedet, *SSAI*
John Berbert, *NASA Goddard Space Flight Center*
Randal Burns, *Johns Hopkins University*
Robert Chadduck, *NARA*
Jack Cole, *US Army Research Laboratory*
Bob Coyne, *IBM*
Jim Finlayson, *Department of Defense*
Dirk Grunwald, *University of Colorado*
Bruce K. Haddon, *Sun Microsystems*
Gene Harano, *National Center for Atmospheric Research*
P C Hariharan, *SES*
Jim Hughes, *StorageTek*
Merritt Jones, *MITRE*
Ben Kobler, *NASA Goddard Space Flight Center*
Steve Louis, *Lawrence Livermore National Laboratory*
Ethan Miller, *University of California, Santa Cruz*
Alan Montgomery, *Department of Defense*
Reagan Moore, *San Diego Supercomputer Center*
Bruce Rosen, *NIST*
Paul Rutherford, *ADIC*
Tom Ruwart, *I/O Performance*
Julian Satran, *IBM Haifa Research Laboratory, Israel*
Donald Sawyer, *NASA Goddard Space Flight Center*
Rodney Van Meter, *Keio University, Japan*

Keynote and Invited Papers
P C Hariharan, *SES*

Tutorial Chair
Jim Hughes, *StorageTek*

Vendor Expo Chair
Gary Sobol, *StorageTek*

Publications Chairs
P C Hariharan, *SES*
Nabil Adam, *Rutgers University*

Work In Progess Chairs
Ethan Miller, *University of California, Santa Cruz*
Randal Burns, *Johns Hopkins University*

Publicity Chairs
Jack Cole, *U.S. Army Research Laboratory*
Sam Coleman, *Lawrence Livermore National Laboratory,* Retired

IEEE Computer Society Liaison
Merritt Jones, *MITRE*

Table of Contents

File Systems - Chair, Curtis Anderson

Site Reports - Chair, Gene Harano

iSCSI and SAN - Chair, Jean-Jacques Bedet

Vendor Solutions - Chair, Bruce Rosen

DATA GRID MANAGEMENT SYSTEMS

Reagan W. Moore, Arun Jagatheesan, Arcot Rajasekar, Michael Wan, Wayne Schroeder
San Diego Supercomputer Center
9500 Gilman Drive, MC 0505
La Jolla, CA 92093
Tel: +1-858-534-5000, Fax: +1- 858-534-5152
e-mail: {moore,arun,sekar,mwan,schroede}@sdsc.edu

Abstract:
The "Grid" is an emerging infrastructure for coordinating access across autonomous organizations to distributed, heterogeneous computation and data resources. Data grids are being built around the world as the next generation data handling systems for sharing, publishing, and preserving data residing on storage systems located in multiple administrative domains. A data grid provides logical namespaces for users, digital entities and storage resources to create persistent identifiers for controlling access, enabling discovery, and managing wide area latencies. This paper introduces data grids and describes data grid use cases. The relevance of data grids to digital libraries and persistent archives is demonstrated, and research issues in data grids and grid dataflow management systems are discussed.

1 Introduction

A major challenge in the design of a generic data management system is the set of multiple requirements imposed by user communities. The amount of data is growing exponentially, both in the number of digital entities and in the size of files. The sources for data are distributed across multiple sites, with data generated in multiple administration domains, and on sites only accessible over wide-area networks. The need for discovery is becoming more important, with data assembled into collections that can be browsed. The need for preservation is becoming more important, both to meet legal data retention requirements, and to preserve the intellectual capital of organizations. In practice, six types of data handling systems are found:

1. Data ingestion systems
2. Data collection creation environments
3. Data sharing environments based on *data grids*
4. Digital libraries for publication of data
5. Persistent archives for data preservation
6. Data processing pipelines

The goal of a generic data management system is to build software infrastructure that can meet the requirements of each of these communities. We will demonstrate that data grids provide the generic data management abstractions needed to manage distributed data, and that all of the systems can be built upon common software.

Data management requires four basic naming conventions (or information categories) for managing data on distributed resources:

1. Resource naming for access to storage systems across administrative domains. This is used to implement data storage virtualization.
2. Distinguished user names for identifying persons across administrative domains. This is used to implement single-sign on security environments.
3. Distinguished file names for identifying files across administrative domains. This is used to implement data virtualization.
4. Context attributes for managing state information generated by remote processes. This is used to implement digital libraries and federate data grids.

A data grid [1,2] provides virtualization mechanisms for resources, users, files, and metadata. Each virtualization mechanism implements a location and infrastructure independent name space that provides persistent identifiers. The persistent identifiers for data are organized as a collection hierarchy and called a "*logical name space*". In practice, the logical name spaces are implemented in a separate metadata catalog for each data grid. Within a data grid, access to data, management of data, and manipulation of data is done via commands applied to the logical name space.

To access data located within another data grid (another logical name space), federation mechanisms are required. To manage operations on the massive collections that are assembled, data flow environments are needed. We illustrate the issues related to data grid creation, data management, data processing, and data grid federation by examining how these capabilities are used by each of the six types of data management systems. We then present the underlying abstraction mechanisms provided by data grids, and close with a discussion of current research and development activities in data flow and federation infrastructure.

2 Data Management Systems

Data management systems provide unifying mechanisms for naming, organizing, accessing, and manipulating context (administrative, descriptive, and preservation metadata) about content (digital entities such as files, URLs, SQL command strings, directories). Each of the types of data management system approaches focuses on a different aspect, and provides specific mechanisms for data and metadata manipulation.

2.1 Data ingestion systems

The Real-time Observatories, Applications, and Data management Network (ROADNet) [3] project manages ingestion of data in real-time from sensors. The data is assembled both synchronously and asynchronously from multiple networks into an object ring buffer (ORB), where it is then registered into a data grid. Multiple object ring buffers are federated into a Virtual Object Ring Buffer (VORB) to support discovery and attribute-based queries. Typical operations are the retrieval of the last ten observations, the tracking of observations about a particular seismic event, and the migration of data from the ORB into a remote storage system.

A second type of data ingestion system is a grid portal, which manages interactions with jobs executing in a distributed environment. The portal provides access to collections for input data, and stores output results back into a collection. A grid portal uses the Grid Security Infrastructure to manage inter-realm authentication between compute and storage sites.

2.2 Data collection creation environments

Scientific disciplines are assembling data collections that represent the significant digital holdings within their domain. Each community is organizing the material into a coherent collection that supports uniform discovery semantics, uniform data models and data formats, and an assured quality. The National Virtual Observatory (NVO) [4] is hosting multiple sky survey image collections. Each collection is registered into a logical name space to provide a uniform naming convention, and standard metadata attributes are used to describe the sky coverage of each image, the filter that was used during observation, the date the image was taken, etc. Collection formation is facilitated by the ability to register the descriptive metadata attributes onto a logical name space. The logical name space serves as the key for correlating the context with each image.

Other examples include: GAMESS [5], computational chemistry data collections of simulation output; and the CEED: Caveat Emptor Ecological Data Repository. Both projects are assembling collections of data for their respective communities.

2.3 Data sharing environments based on data grids

Scientific disciplines promote the sharing of data. While collections are used to organize the content, data grids are used to manage content that is distributed across multiple sites. The data grid technology provides the logical name space for registering files, the inter-realm authentication mechanisms, the latency management mechanisms, and support for high-speed parallel data transfers. An example is the Joint Center for Structural Genomics data grid which generates crystallographic data at the Stanford Linear Accelerator and pushes the data to SDSC for storage in an archive and analysis of protein structures. The Particle Physics Data Grid [6] federates collections housed at Stanford and Lyon, France for the BaBar high energy physics experiment [7]. The Biomedical Informatics Research Network (BIRN) [8] is using a data grid to share data from multiple Magnetic Resonance Imaging laboratories. Each project implements access controls that are applied on the distributed data by the data grid, independently of the underlying storage resource.

2.4 Digital libraries for publication of data

An emerging initiative within digital libraries is support for standard digital reference sets that can be used by an entire community. The standard digital reference sets are created from observational data collections, or from simulations, and are housed within the digital library. Curation methods are applied to assure data quality. Discovery mechanisms are supported for attribute-based query. An example is the HyperAtlas catalog that is being created for the 2MASS and DPOSS astronomy sky surveys [4,19,20]. The catalog projects each image to a standard reference frame, organizes the

projections into an atlas of the sky, and supports discovery through existing sky catalogs of stars and galaxies.

The organizations participating in a collaboration may share their digital entities using a collective logical view. Multiple logical views could be created for the same set of distributed digital entities. These logical views may be based on different taxonomies or business rules that help in the categorization of the data. The organizations, apart from sharing the physical data, could also share the logical views as publication mechanisms.

The library community applies six data management processes to digital entities:

1. Collection building to organize digital entities for access
2. Content management to store each digital image
3. Context management to define descriptive metadata
4. Curation processes to validate the quality of the collection
5. Closure analyses to assert completeness of the collection and the ability to manipulate digital entities within the collection
6. Consistency processes to assure that the context is updated correctly when operations are performed on the content.

2.5 Persistent archives for data preservation

The preservation community manages archival collections for time periods that are much longer that the lifetimes of the underlying infrastructure [9,10,11,25,26,27]. The principal concern is the preservation of the authenticity of the data, expressed as an archival context associated with each digital entity, and the management of technology evolution. As new more cost effective storage repositories become available, and as new encoding formats appear for data and metadata, the archival collection is migrated to the new standard. This requires the ability to make replicas of data on new platforms, provide an access abstraction to support new access mechanisms that appear over time, and migrate digital entities to new encoding formats or emulate old presentation applications. Example projects include a persistent archive for the National Science Digital Library, and a persistent archive prototype for the National Archives and Records Administration [12.13]. Both projects manage data that is stored at multiple sites, replicate data onto different types of storage repositories, and support both archival and digital library interfaces.

The preservation community applies their own standard processes to data:

1. Appraisal – the decision for whether a digital entity is worth preserving
2. Accession – the controlled process under which digital entities are brought into the preservation environment
3. Arrangement – the association of digital entities with a record group or record series
4. Description – the creation of an archival context specifying provenance information
5. Preservation – the creation of an archival form for each digital entity and storage
6. Access – the support for discovery services

2.6 Data Processing Pipelines

The organization of collections, registration of data into a data grid, curation of data for ingestion into a digital library, and preservation of data through application of archival processes, all need the ability to apply data processing pipelines. The application of processes is a fundamental operation needed to automate data management tasks. Scientific disciplines also apply data processing pipelines to convert sensor data to a standard representation, apply calibrations, and create derived data products. Examples are the Alliance for Cell Signaling digital library, which applies standard data analysis techniques to interpret each cell array, and the NASA Earth Observing Satellite system that generates derived data products from satellite observations.

Data processing pipelines are also used to support knowledge generation. The steps are:

1. Apply semantic labels to features detected within a digital entity
2. Organize detected features for related digital entities within a collection
3. Identify common relationships (structural, temporal, logical, functional) between detected features
4. Correlate identified relationships to physical laws

3 Generic requirements

Multiple papers that summarize the requirements of end-to-end applications have been generated in the Global Grid Forum [14]. They range from descriptions of the remote operations that are needed to manage large collections, to the abstraction mechanisms that are needed for preservation. The capabilities can be separated into four main categories: Context management, data management, access mechanisms, and federation mechanisms. The capabilities are provided by data grids:

Context management mechanisms

Global persistent identifiers for naming files.

Organization of context as collection hierarchy

Support for administrative metadata to describe the location and ownership of files

Support for descriptive metadata to support discovery through query mechanisms

Support for browsing and queries on metadata

Information repository abstraction for managing collections in databases

Data management mechanisms

Storage repository abstraction for interacting with multiple types of storage systems

Support for the registration of files into the logical name space

Inter-realm authentication system for secure access to remote storage systems

Support for replication of files between sites

Support for caching onto a local storage system and for accessing files in an archive

Support for aggregating files into containers

Access mechanisms

Standard access mechanisms: Web browsers, Unix shell commands, Windows browsers, Python scripts, Java, C library calls, Linux I/O redirection, WSDL, etc.

Access controls and audit trails to control and track data usage

Support for the execution of remote operations for data sub-setting, metadata extraction, indexing, third-party data movement, etc.

Support for bulk data transfer of files, bulk metadata transfer, and parallel I/O

Federation mechanisms

Cross-registration of users between data grids

Cross-registration of storage resources data grids

Cross-registration of files between data grids

Cross-registration of context between data grids

Data Grids provide transparency and abstraction mechanisms that enable applications to access and manage data as if they were local to their home system. Data grids are implemented as federated client-server middleware that use collections to organize distributed data.

4 Data Grid Implementation

An example of a data grid is the Storage Resource Broker (SRB) from the San Diego Supercomputer Center [15,16,17]. The SRB manages context (administrative, descriptive, and preservation metadata) about content (digital entities such as files, URLs, SQL command strings, directories). The content may be distributed across multiple types of storage systems across independent administration domains. By separating the context management from the content management, the SRB easily provides a means for managing, querying, accessing, and preserving data in a distributed data grid framework. Logical name spaces describe storage systems, digital file objects, users, and collections. Context is mapped to the logical name spaces to manage replicas of data, authenticate users, control access to documents and collections, and audit accesses. The SRB manages the context in a Meta data Catalog (MCAT) [18], organized as a collection hierarchy. The SRB provides facilities to associate user-defined metadata, both free-form attribute-based metadata and schema-based metadata at the collection and object level and query them for access and semantic discovery. The SRB supports queries on descriptive attributes [21]. The SRB provides specific features needed to implement digital libraries, persistent archive systems [10,11,12,13] and data management systems [22,23,24].

The Storage Resource Broker (SRB) in earlier versions used a centralized MCAT [18] for storing system-level and application-level metadata. Though essential for consistent operations, the centralized MCAT poses a problem. It can be considered a single-point of failure as well as a potential bottleneck for performance. Moreover, when users are widely distributed, users remote from the MCAT may see latency unacceptable for interactive data access.

In order to mitigate these problems, the SRB architecture has been extended to a federated environment, called zoneSRB. The ZoneSRB architecture provides a means for multiple context catalogs to interact with each other on a peer-to-peer basis and synchronize their data and metadata. Each zoneSRB system can be autonomous, geographically distant, and administer a set of users, resources and data that may or may

not be shared by another zoneSRB. Each zoneSRB has its own MCAT for providing the same level of features and facilities as done by the older SRB system.

The main advantage of the zoneSRB system is that now, there is no single point of failure, as multiple MCATs can be federated into a multi-zoneSRB system. Users can be distributed across the zones to improve quality of performance and minimize access latencies to geographically distant metadata catalogs. The multiple zoneSRBs can share metadata and data based on policies established by the collaborating administrators. The level of collaboration can be varied to specify how much of the information is shared, partitioned or overlapped, and whether the interactions are controlled by the users or the zone administrators.

More information about the SRB can be found in [15,16,17,18]. In a nutshell, the SRB provides all of the capabilities listed as generic requirements. The SRB provides interoperability mechanisms that map users, datasets, collections, resources and methods to global namespaces. It also provides abstractions for data management functionality such as file creation, access, authorization, user authentication, replication and versioning, and provides a means to associate metadata and annotation with data objects and collections of data objects. Descriptive metadata is used for searching at the semantic level and discovery of relevant data objects using the attribute-based discovery paradigm. Figure 1 provides details about the modules that form the core of the SRB services.

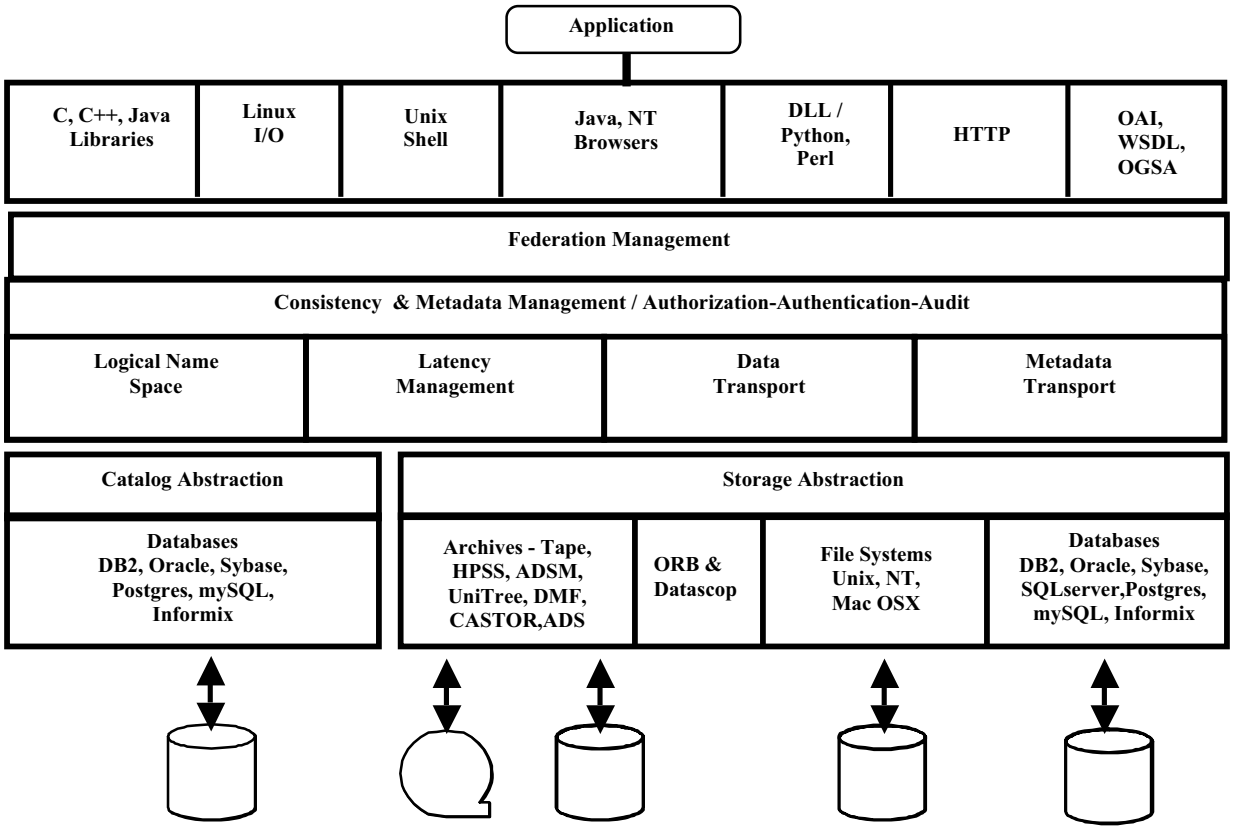


Figure 1. Storage Resource Broker

The SRB server is built using three layers. The top-layer is a set of server access APIs written in C and Java that are used to provide multiple client interfaces. The middle layer provides the intelligence for collection-based management, federation of data grids, data movement, authentication, and authorization. The bottom-layer is a set of storage drivers and MCAT database drivers that are used to connect to diverse resources. These drivers have a well-defined and published API such that a new storage resource or data server system can be integrated easily into the SRB system. For example, custom-based interfaces to special drivers such as the Atlas Data Store System and CERN's CASTOR storage systems were written in just a few days. The SRB drivers for storage include, HPSS, ADSM, Unix/Linux /Mac OSX and NT file systems, and for databases includes DB2, Oracle, Postgres, Informix, and Sybase.

The middle layer of the SRB system supports the federation consistency and control mechanisms needed to integrate multiple data grids and encapsulates much of the intelligence needed to manage a data grid. This layer interacts with the MCAT to access the context needed to control each type of data service.

The data grid technology based on the SRB that is in production use at SDSC at this date manages over 90 Terabytes of data comprising over 16 million files.

5 Data Grid Federation

The specification of appropriate federation mechanisms is still a research project. The federation of multiple data grids basically imposes constraints on the cross-registration of users, resources, files, and context. The constraints may be invoked by either a user or automatically implemented by the data management system. The constraints may be set for either no cross-registration, partial cross-registration, or complete cross-registration. It is possible to identify over 1500 possible federation approaches by varying the type of cross-registration constraints that are imposed. In practice, ten of the approaches are either in active use or proposed for use within scientific projects. Each data grid is called a zone, with its own metadata catalog managing the logical name spaces for its users, resources, files, and context. The approaches include:

5.1 Occasional Interchange

This is the simplest model in which two or more zones operate autonomously with very little exchange of data or metadata. The two zones exchange only user-ids for those users who need access across zones. Most of the users stay in their own zone accessing resources and data that are managed by their zone MCAT. Inter-zone users will occasionally cross zones, browsing collections, querying metadata and accessing files for which they have read permission. These users can store data in remote zones if needed but these objects are only accessible to users in the other zones. This model provides the greatest degree of autonomy and control. The cross-zone user registration is done not for every user from a zone but only for selected users. The local SRB administrator controls who is given access to the local SRB system and can restrict these users from creating files in the local SRB resources. (NPACI driven federation model [28])

5.2 **Replicated Catalog**

In this model, even though there are multiple MCATs managing independent zones, the overall system behaves as though it were a single zone with replicated MCATs. Metadata about the tokens being used, users, resources, collections, containers and data objects are all synchronized between all MCATs. Hence, the view from every zone is the same. An object created in a zone is registered as an object in all other sister zones and any associated metadata is also replicated. This model provides a completely replicated system that has a high degree of fault-tolerance for MCAT failures. The user can still access data even if their local MCAT becomes non-functional. The degree of synchronization, though very high in principle, in practice is limited. The MCATs may be out of synchronization on newly created data and metadata. The periodicity of synchronization is decided by the cooperating administrators and can be as long as days if the systems change slowly. An important point to note is that because of these delayed synchronizations, one might have occasional logical name clashes. For example, a data object with the same name and in the same collection might be created in two zones almost at the same time. Because of delayed synchronization both will be allowed in their respective zones. But when the synchronization is attempted, the system will see a clash when registering across zones. The resolution of this has to be done by mutual policies set by the cooperating administrators. In order to avoid such clashes, policies can be instituted with clear lines of partitioning about where one can create a new file in a collection. (NARA federation model [12])

5.3 **Resource Interaction**

In this model resources are shared by more than one zone and hence they can be used for replicating data. This model is useful if the zones are electronically distant, but want to make it easier for users in the sister zone to access data that might be of mutual interest. A user in a zone replicates data into the shared resources (either using synchronous replication or asynchronous replication as done in a single zone). Then the metadata of these replicated objects is synchronized across the zones. User names need not be completely synchronized. (BIRN federation model [8])

5.4 **Replicated Data Zones**

In this model two or more zones work independently but maintain the same data across zones, i.e., they replicate data and related metadata across zones. In this case, the zones are truly autonomous and do not allow users to cross zones. In fact, user lists and resources are not shared across zones. But data stored in one zone is copied into another zone along with related metadata, by a user who has accounts in the sister zones. This method is very useful when two zones are operating across a wide-area network has to share data and the network delay in accessing data across the zones has to be reduced. (BaBar federation model [7])

5.5 **Master-Slave Zones**

This is a variation of the 'Replicated Data Zones' model in which new data is created at a Master site and the slave sites synchronize with the master site. The user list and resource list are distinct across zones. The data created at the master are copied over to the slave zone. The slave zone can create additional derived objects and metadata but these may

not be shared back to the Master Zone. (PDB federation model)

5.6 Snow-Flake Zones

This is a variation of the 'Master-Slave Zones' model, One can view this as a hierarchical model, where a Master Zone creates the data that is copied to the slave zones, whose data in turn gets copied into other slave zones lower in the hierarchy. Each level of the hierarchy can create new derived products of data and metadata, can have their own client base, and can choose to propagate only a subset of their holdings to their slave zones. (CMS federation model [23]).

5.7 User and Data Replica Zones

This is another variation of the 'Replicated Data Zones' where not just the data get replicated but also user names are exchanged. This model allows users to access data in any zone. This model can be used for wide-area enterprises where users travel across zones and would like to access data from their current locations for improved performance. (Web cache federation model)

5.8 Nomadic Zones - SRB in a Box

In this model, a user might have a small zone on a laptop or other desktop systems that are not always connected to other zones. The user during his times of non-connectedness can create new data and metadata. The user on connecting to the parent zone will then synchronize and exchange new data and metadata across the user-zone and the parent zone. This model is useful for users who have their own zones on laptops. It is also useful for zones that are created for ships and nomadic scientists in the field who periodically synchronize with a parent zone. (SIOExplorer federation model)

5.9 Free-floating Zones – myZone

This is a variation of the 'Nomadic Zone' model having multiple stand-alone zones but no parent zone. These zones can be considered peers and possibly have very few users and resources. These zones can be seen as isolated systems running by themselves (like a PC) without any interaction with other zones, but with a slight difference. These zones occasionally "talk" to each other and exchange data and collections. This is similar to what happens when we exchange files using zip drives or CDs or as occasional network neighbors. This system has a good level of autonomy and isolation with controlled data sharing. (Peer-to-peer or Napster federation model)

5.10 Archival Zone, BackUp Zone

In this model, there can be multiple zones with an additional zone called the archive. The main purpose of this is to create an archive of the holdings of the first set of zones. Each zone in the first set can designate the collections that need to be archived. This provides for backup copies for zones which by themselves might be fully running on spinning disk. (SDSC backup federation model, NASA backup federation model [29])

6 Grid Dataflow

A second research area is support for data flow environments, in which state information is kept about the processing steps that have been applied to each digital entity in a work set.

6.1 Need for peer-to-peer Data Grid Dataflows

A dataflow executes multiple tasks. Each task might require: different resources; access to different data collections for input; storage of output products onto physically distributed resources within a data grid; and disparate services that might be in the form of web/grid services or simply executables of an application. The dataflow is described in a data grid language. The dataflow is executed through a dataflow engine. Each dataflow engine needs to be able to communicate with other dataflow engines in a peer-to-peer federation for coordination. This allows dynamic distributed execution, without having to specify a pre-planned schedule.

Placement scheduling is still required to find the right location for execution of each task. In the data grid, the tasks in the dataflow could be executed in any of distributed resources within the participating administrative domains. In general the following factors must be considered for dataflow scheduling:

Appropriateness of a given resource for a particular task: Is there enough disk space to hold result sets, and is the compute resource powerful enough to execute the task within a desired time? Are the tasks sufficiently small that they could be processed by less powerful systems?

Management of data movement: How can the amount of data moved for both input and output files and for the executable be minimized?

Co-location of dependent tasks: How can tasks be co-located on the same administrative domain or resource to minimize coordination messages that have to be sent across the network?

6.2 Grid Dataflow System Requirements

Collections of data sets can be manipulated in a Data Grid dataflow. Instead of creating a separate dataflow for each file, state information can be maintained about the aggregated set of files for which processes have been applied. Related issues are:

Management of processing state: (e.g.) What information needs to be maintained about each process?

Control procedures: (e.g.) What types of control mechanisms are needed to support loops over collections?

Dynamic status queries: (e.g.) Can a process detect the state of completion of other processes through knowledge of the placement schedule?

6.3 Data Grid Language

The SDSC Matrix project [33], funded by the NSF Grid Physics Network (GriPhyN) [30], NIH Biomedical Informatics Research Network (BIRN) [8] and NSF Southern California Earthquake Center (SCEC) [31], has developed a data grid language to describe grid dataflow. Just like SQL (Structured Query Language) is used to interact with the databases, the Data Grid Language (DGL) is used to interact with the data grids

and dataflow environments. DGL is XML-based and uses a standard schema that describes:

Control-based dataflow structures. These include sequential, parallel, and aggregated process execution.

Context-based dataflow structures. These include barriers (synchronization points or milestones), "For loops" (iteration over task sets) and "For Each loops" (iteration over collection of files).

Event Condition Alternate Action (ECA) rules. Any event in the workflow engine like completion or start of a task could be used to trigger a condition to be evaluated dynamically and execute any of the alternate dataflow actions. The conditions could be described using XQuery or any other language that would be understood by the dataflow engine. This allows other useful or simple workflow query languages to be used along with DGL.

Variables. Both global variables and local variables can be managed for the dataflow. The variables are related to the dataflow, rather than an individual file that is manipulated by the dataflow. Hierarchical scoping is used to restrict the use of the dataflow variables to aggregates of processes.

Discovery. Queries from external grid processes are supported for determining the completion status of a process and the state of variables.

A simple example of the use of dataflow systems is the management of the ingestion of a collection into a data grid. The SCEC project implemented the collection ingestion as a dataflow using the data grid language and executed the dataflow using a SDSC Matrix Grid workflow engine.

6.4 SDSC Matrix Architecture

The architecture of the SDSC Matrix dataflow engine is shown in Figure 2. The components are layered on top of agents that can execute either SRB or other processes (SDSC Data Management Cyberinfrastructure, java classes, WSDL services and other executables). The matrix dataflow engine tracks the dataflow execution. The dataflow execution state can be queried by other applications or other dataflows that are executed by the matrix engine. Persistence of the dataflow execution state is held in memory and exported to a relational database.

Clients send DGL dataflow requests as SOAP [32] messages to the Java XML (JAXM) messaging interface (Fig 2). The Matrix web service receives these SOAP messages and forwards the DGL requests to the Data Grid Request Processor. The Request Processor parses the DGL requests, which could be either a *data grid transaction* (new dataflow) or a *status query* on another dataflow. A data grid transaction request is a long running dataflow involving the execution of multiple data management and compute intensive processes. The *Transaction Handler* registers a new transaction and starts the book keeping and execution of the processes. The *Status Query Handler* is used to query the state of execution of the transaction and the variables associated with a dataflow. In Figure 2, the Matrix Engine components shown in white boxes have been implemented for a stand-alone matrix workflow engine (version 3.1). Those in solid (black) boxes are a work in progress to provide peer-to-peer grid workflow and involve protocols to distribute the workflow. The P2P broker will be based on Sangam protocols [34] to

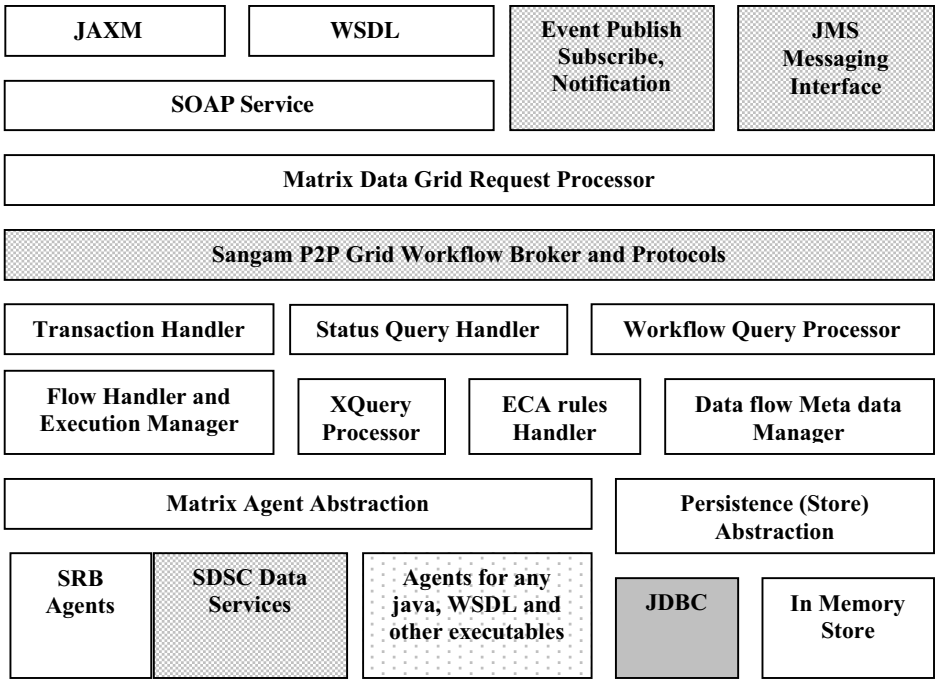


Figure 2. SDSC Matrix Grid Workflow Engine Architecture for data flows

facilitate Peer-to-peer brokering between workflow engines. The protocols will be loosely coupled with resource scheduling algorithms.

7 Conclusion

Data grids provide a common infrastructure base upon which multiple types of data management environments may be implemented. Data grids provide the mechanisms needed to manage distributed data, the tools that simplify automation of data management processes, and the logical name spaces needed to assemble collections. The Storage Resource Broker data grid is an example of a system that has been successfully applied to a wide variety of scientific disciplines for management of massive collections. Current research issues include identification of the appropriate approaches for federating data grids, and the development of capable data flow processing systems for the management of data manipulation.

8 Acknowledgement

The results presented here were supported by the NSF NPACI ACI-9619020 (NARA supplement), the NSF NSDL/UCAR Subaward S02-36645, the NSF Digital Library Initiative Phase II Interlib project, the DOE SciDAC/SDM DE-FC02-01ER25486 and DOE Particle Physics Data Grid, the NSF National Virtual Observatory, the NSF Grid Physics Network, and the NASA Information Power Grid. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, the National Archives and Records Administration, or the U.S. government.

The authors would also like acknowledge other members of the SDSC SRB team who have contributed to this work including: George Kremenek, Bing Zhu, Sheau-Yen Chen, Charles Cowart, Roman Olschanowsky, Vicky Rowley and Lucas Gilbert. The members

of the SDSC Matrix Project include Reena Mathew, Jon Weinberg, Allen Ding and Erik Vandekieft.

9 References

[1] Foster, I., and Kesselman, C., (1999) "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann.

[2] Rajasekar, A., M. Wan, R. Moore, T. Guptill, "Data Grids, Collections and Grid Bricks," Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies, April 7-10, 2003, San Diego, USA.

[3] Real-time Observatories, Applications, and Data management Network (RoadNet), http://roadnet.ucsd.edu/

[4] NVO, (2001) "National Virtual Observatory", (http://www.srl.caltech.edu/nvo/).

[5] GAMESS "General Atomic Molecular Electronic Structure Systems - Web Portal. (https://gridport.npaci.edu/GAMESS/).

[6] PPDG, (1999) "The Particle Physics Data Grid", (http://www.ppdg.net/, http://www.cacr.caltech.edu/ppdg/).

[7] BABAR Collaboration (B. Aubert et al.)., "The First Year of the Babar Experiment At PEP-II. 30th International Conference on High-Energy Physics (ICHEP 2000), Japan.

[8] BIRN, "The Biomedical Informatics Research Network", http://www.nbirn.net

[9] Rajasekar, A., R. Marciano, R. Moore, (1999), "Collection Based Persistent Archives," Proceedings of the 16th IEEE Symposium on Mass Storage Systems, 1999.

[10] Moore, R., C. Baru, A. Rajasekar, B. Ludascher, R. Marciano, M. Wan, W. Schroeder, and A. Gupta, (2000), "Collection-Based Persistent Digital Archives – Parts 1& 2", D-Lib Magazine, April/March 2000, http://www.dlib.org/

[11] Moore, R., A. Rajasekar, "Common Consistency Requirements for Data Grids, Digital Libraries, and Persistent Archives", Grid Protocol Architecture Research Group draft, Global Grid Forum, April 2003.

[12] US National Archives and Records Administration, http://www.archives.gov/, also see http://www.sdsc.edu/NARA/

[13] Moore, R., C. Baru, A. Gupta, B. Ludaescher, R. Marciano, A. Rajasekar, (1999), "Collection-Based long-Term Preservation," GA-A23183, report to National Archives and Records Administration, June, 1999.

[14] GGF, "The Global Grid Forum" (http://www.ggf.org/)

[15] SRB, "Storage Resource Broker Website", SDSC (http://www.npaci.edu/dice/srb).

[16] Rajasekar, A., Wan, M., Moore, R.W., Schroeder, W., Kremenek, G., Jagatheesan, A., Cowart, C., Zhu, B., Chen, S.Y. and Olschanowsky, R., "Storage Resource Broker – Managing Distributed Data in a Grid," *Computer Society of India Journal, special issue on SAN*, 2003

[17] Rajasekar, A., Wan, M., Moore, R.W., Jagatheesan, A. and Kremenek, G., "Real Experiences with Data Grids – Case-studies in using the SRB," Proceedings of *6th International Conference/Exhibition on High Performance Computing Conference in Asia Pacific Region (HPC-Asia)*, December 2002, Bangalore, India

[18] MCAT - "The Metadata Catalog", http://www.npaci.edu/DICE/SRB/mcat.html

[19] 2-Micron All Sky Survey (2MASS), http://www.ipac.caltech.edu/2mass/

[20] Digital Palomar Observatory Sky Survey, http://www.astro.caltech.edu/~george/dposs/

[21] Moore R. and A. Rajasekar, "Data and Metadata Collections for Scientific Applications," High Performance Computing and Networking, Amsterdam, NL, 2001.
[22] Wan, M., A. Rajasekar, R. Moore, "A Simple Mass Storage System for the SRB Data Grid," Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies, April 7-10, 2003, San Diego, USA.
[23] Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., and Stockinger, K. (2000) "Data Management in an International Data Grid Project," *IEEE/ACM International Workshop on Grid Computing Grid'2000*, Bangalore, India 17-20 December 2000. (http://www.eu-datagrid.org/grid/papers/data_mgt_grid2000.pdf).
[24] Moore, R., C. Baru, A. Rajasekar, R. Marciano, M. Wan: Data Intensive Computing, In ``The Grid: Blueprint for a New Computing Infrastructure'', eds. I. Foster and C. Kesselman. Morgan Kaufmann, San Francisco, 1999.
[25] Thibodeau, K., "Building the Archives of the Future: Advances in Preserving Electronic Records at the National Archives and Records Administration", U.S. National Archives and Records Administration, http://www.dlib.org/dlib/february01/thibodeau/02thibodeau.html
[26] Underwood, W. E., "As-Is IDEF0 Activity Model of the Archival Processing of Presidential Textual Records," TR CSITD 98-1, Information Technology and Telecommunications Laboratory, Georgia Tech Research Institute, December 1, 1988.
[27] Underwood, W. E., "The InterPARES Preservation Model: A Framework for the Long-Term Preservation of Authentic Electronic Records". Choices and Strategies for Preservation of the Collective Memory, Toblach/Dobbiaco Italy 25-29 June 2002, Archivi per la Storia.
[28] NPACI Data Intensive Computing Environment thrust, http://www.npaci.edu/DICE/
[29] NASA Information Power Grid (IPG), (http://www.ipg.nasa.gov/)
[30] GriPhyN, "The Grid Physics Network", (http://www.griphyn.org/).
[31] SCEC Web Site, Southern California Earthquake Center, (http://www.scec.org/)
[32] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., and Winer, D., *"Simple Object Access Protocol (SOAP)"* W3C Note. http://www.w3.org/TR/SOAP/
[33] SDSC Matrix Project Web site http://www.npaci.edu/DICE/SRB/matrix/
[34] Jagatheesan, A., "Architecture and Protocols for Sangam Communities and Sangam E-Services Broker," Technical Report, (Master's Thesis) CISE Department, University of Florida, 2001 http://etd.fcla.edu/UF/anp1601/ArunFinal.pdf
[35] Helal, A., Su, S.Y.W., Meng, Jei, Krithivasan, R. and Jagatheesan, R., "The Internet Enterprise," Proceedings of Second IEEE/IPSJ Symposium on Applications and the Internet (SAINT 02), February 2002, Japan. http://www.harris.cise.ufl.edu/projects/publications/Internet_Enterprise.pdf

LONG-TERM STEWARDSHIP OF GLOBALLY-DISTRIBUTED REPRESENTATION INFORMATION

David Holdsworth
Information Systems Services
Leeds University
LS2 9JT UK
+44 113 343 5401
e-mail: ecldh@leeds.ac.uk

Paul Wheatley
Edward Boyle Library
Leeds University
LS2 9JT UK
+44 113 343 5830
e-mail: P.R.Wheatley@leeds.ac.uk

Background

Leeds was a major participant in three projects looking at digital preservation, viz Cedars [1] (jointly with the Universities of Oxford and Cambridge), CAMiLEON [2] (jointly with the University of Michigan), and the Representation and Rendering Project [3]. With this background, work is beginning on setting up a digital curation centre [4]. for UK academia.

As a result of this work, we strongly favour a policy of retaining the original byte-stream (or possibly bit-stream, see below) as the master copy, and evolving representation information (including software tools) over time to guarantee continued access to the intellectual content of the preserved material. This paper attempts to justify that approach, and to argue for its technical feasibility and economic good sense.

Thus we need long-term stewardship of the byte-streams, and long-term stewardship of the representation information. We use the term *representation information* in the sense of the OAIS model [5]. The purpose of the representation information is to give future access to the intellectual content of preserved byte-streams. Without stewardship of the representation information we would not be exercising stewardship of the preserved data.

Inevitability of Change in the context of long-term

Since computers were invented in the 1940s and 50s, there have many changes in the representation of data. The binary digit has survived as an abstraction, and in today's world the byte is a world-wide standard, although we sometimes have to call it an *octet*.

All we can be certain of for the long-term future is that there will be further change. However, even though the technology used for representing such bits and bytes has

changed over time, the abstract concept lives on. Nonetheless, the uses to which those bits and bytes can be put have grown massively over the years.

Our work has always taken the view that "long-term" means many decades. As digital information technology is barely 60 years old, and we have already lost all of the software from the earliest machines, we need to mend our ways. We should plan that our digital information will still be safe and accessible in 100 years. It is then likely that developments over that time will render the material safe for millennia. In short, we are talking of a time span over which all of our existing hardware technology is likely to be obsolete, and also much of the software.

It is the representation information that makes the bridge between IT practices at the time of preservation, and IT practices at the time of access to the information.

Abstraction is Vital

We can be confident that the concept of information will survive the passage of time, and even the concept of digital information. We need to bridge the longevity of the information concept to the certain mortality of the media on which the data lives. Our approach is to ensure that everything is represented as a sequence of bytes. We have confidence that the ability to store a sequence of bytes will survive for many decades, and probably several centuries. Current technology usually does this by calling this sequence a file, and storing it in a file system. There are many files in today's computer systems that had their origins in previous systems.

The challenge that remains is to maintain the ability to extract the information content of such byte-streams. The knowledge of the formats of such preserved data is itself information, and is amenable to being represented digitally, and is thus amenable to preservation by the same means as we use for the data itself.

By taking this focus on the storage of a stream of bytes, we divide the problem into two.

1. Providing media for storage, and copying byte-streams from older technology to newer technology.
2. Maintaining knowledge of the data formats, and retaining the ability to process these data formats in a cost-effective manner.

The OAIS representation net is the means by which the knowledge is retained. By treating all data as an abstract byte-stream at the lowest level, we have a common frame of reference in which we can record representation information, independent of any particular data storage technology, and any particular data storing institution. We have a framework in which representation information will be globally relevant.

Keep the Original Data

We have no faith in long-lived media [6]. Our approach is always to keep the original data as an abstract byte-stream and to regard it as the master.

Why? Because it is the only way to be sure that nothing is lost. Format conversion can lose data through moving to a representation incapable of handling all the properties of the original. It can also lose data through simple software error in the conversion process that goes undetected until it is too late to read the previous data.

One of us (DH) has personal experience of both situations. One in which the data was damaged, and one in which potential damage was avoided by keeping the original and producing a format conversion tool.

How? We certainly cannot preserve the medium upon which the data is stored. In Cedars we developed the concept of an *underlying abstract form* which enabled us to convert any digital object into a byte-stream from which we could regenerate the *significant properties* of the original. Our approach is to preserve this byte-stream indefinitely, copying it unchanged as storage technology evolves.

The question then remains as to how we continue to have *access to the intellectual content* (another Cedars phrase) of the data, and not merely a stream of bytes. Our answer to this is that we evolve the representation information over time so that it provides us with the means to transform our original into a form that can be processed with the tools current at the time of access. We believe that our work in the CAMiLEON project has shown this to be feasible in the case of a very difficult original digital object of great historical importance. Using emulation we successfully preserved the accessibility of the BBC's "Domesday" project, see below and [16].

The very essence involves identifying appropriate abstractions, and then using them as the focus of the rendering software. We achieve longevity by arranging that the rendering software is implemented so as remain operational over the decades. The application of our approach to emulation is covered in *Emulation, Preservation and Abstraction* [7]. We have also investigated the same technique of retention of the original binary data coupled with evolving software tools in the context of format migration [8].

Format Conversion — when?

It is obvious that when data is to be accessed some time after its initial collection, the technology involved in this access will differ markedly from that in use when data collection took place. There is also the real possibility that other technologies have been and gone in the interim. Thus, format conversion is inevitable.

For data held in currently common formats, the amount of representation information needed is trivial. Meaningful access to the data normally happens at the click of a mouse.

A current computer platform will render a PDF file merely by being told that the format is PDF. Conversely, faced with an EBCDIC file of IBM SCRIPT mark-up, the same current platform might well render something with little resemblance to the original, whereas back in 1975, the file could be rendered as formatted text with minimal formality.

However, if we have representation information for IBM SCRIPT files that points us at appropriate software for rendering the file contents on current platforms, the historic data becomes accessible to today's users. Alternatively, we could have converted all the world's IBM SCRIPT files into Word-for-Windows, or LaTeX, or .... We could argue about the choice until all the current formats become obsolete, and we could well have chosen a format that itself quickly became obsolete. We could have been tempted to convert from EBCDIC to ASCII, but that could have lost information because EBCDIC has a few more characters than ASCII.

We recommend that the format of preserved data be converted only when access is required to the data, i.e. on creation of *the Dissemination Information Package* (DIP). For a popular item, it would obviously make sense to cache the DIP, but not to allow the reformatted DIP to replace the original as master. This means that the tracking of developments in storage technology involves only the copying of byte-streams. Moreover, when the format conversion has to be done, there will be improved computational technology with which to do it [9].

Indirection is Vital

> There isn't a problem in computer science that cannot be solved by an extra level of indirection. *Anon*

The essence of our approach involves keeping the preserved data unchanged, and ensuring that we always have representation information that tells us how to access it, rather than repeatedly converting to a format in current use. We take the view that it is very difficult (impossible?) to provide representation information that will be adequate for ever. We propose that representation information evolves over time to reflect changes in IT practice. This clearly implies a structure in which each stored object contains a pointer to its representation information. This is easily said, but begs the question as to the nature of the pointer.

We need a pointer that will remain valid over the long-term (i.e. 100 years). We need to be wary of depending on institutions whose continued existence cannot be guaranteed.

Alongside this need for a pointer, we also have a need for a reference ID for each preserved object. This needs to be distinct from the location of the object, but there needs to be a service that translates a reference ID into a current location. This is the essence of the Cedars architecture [10].

Reference IDs could be managed locally within an archive store. Such IDs could then be made global, by naming each archive store, and prefixing each local name with that of the archive store.

There are various global naming schemes, ISBN, DNS, Java packages, URL, URI, URN, DOI, etc. It may even be necessary to introduce another one, just because there is no clear long-term survivor. What is certain is that there have to be authorities that give out reference IDs and take responsibility for translating these IDs into facilities for access to the referenced stored objects.

If we grasp the nettle of a global name space for reference IDs of stored objects and keep the representation information in the same name space, we have the prospect of sharing the evolving representation information on a world-wide basis. This will imply some discipline if dangling pointers are to be avoided.

Enhance Representation Nets over time

In the Cedars Project we produced a prototype schema for a representation net following the OAIS model, and populated it with some examples. After this experience, we had some new ideas on the schema of the representation net. We believe that it is inevitable that this area is allowed to develop further, and that operational archives are built so that evolution in this area is encouraged to take place. We must accept that there is likely to be revision in the OAIS model itself over the 100 year time-frame.

Also, we could see that to require a fully specified representation net before allowing ingest could act as a disincentive to preservation of digital objects whose value is not in doubt. In many cases, representation information existed as textual documentation. An operational archive needs to be capable of holding representation information in this purely textual form, although with an ambition to refine it later. Such information would not actually violate the OAIS model, but there is a danger of being over-prescriptive in implementing the model. For instance the NISO technical metadata standard for still images [11] has over 100 elements, at least half of which are compulsory.

For some formats the most useful representation information is in the form of viewing software. We need our representation nets to enable the discovery of such software (see below). Many current objects need only to be introduced to a typical desktop computer in order for them to be rendered. On the other hand, we experimented with obsolete digital objects (from 1970s and 1980s) in order to see some of the issues likely to arise when our grandchildren wish to gain access to today's material. We even tried to imagine how we would have gone about preserving for the long-term future using the technology of the 1970s. It was abundantly clear that ideas are very different now than they were 30 or 40 years ago. We must expect that today's ideas could well be superseded over the long-term.

In order to accommodate this, we must allow the content of objects in the representation net to be changed over time, in sharp contrast to the original preserved objects where we

are recommending retention of original byte-streams. It is vital that the reference ID that is originally used for representation information is re-used for newer representation information which gets produced as a result of development of new tools and ideas. That way, old data gets to benefit from new techniques available for processing it. The representation information that is being replaced should of course be retained, but with a new ID, which should then be referenced by the replacement.

Representation Nets should link to software

Our representation nets in Cedars very deliberately contained software, or in some cases references to it. We have no regrets on this issue. Ideally we want software in source form in a programming language for which implementations are widely available, but it seems churlish to refuse to reference the Acrobat viewer as a way of rendering PDF files, just because we do not have the source, but see example 1 below.

A format conversion program that is known to work correctly on many different data objects is clearly a valuable resource for access to the stored data, and should be available via the representation network.

As regards the issue of longevity of such software, we argued earlier for the longevity of abstract concepts such as bits, bytes and byte-streams. Programming languages are also abstract concepts, and they too can live for a very long time. Current implementations of C or FORTRAN will run programs from long ago. Other languages which have been less widely used also have current implementations that function correctly.

The source text of a format conversion program which is written in a language for which no implementation is available is still a valuable specification of the format, and has the benefit of previously proven accuracy. We address the issue of evolving emulator programs in *C-ing Ahead for Digital Longevity* [12], which proposes using a subset of C as the programming language for writing portable emulators.

Examples

We illustrate the way in which we see representation information evolving over time, by reference to three examples drawn from rather different computational environments.

Example 1: Acrobat files

In today's IT world it is very common to use Adobe Acrobat® portable document format (PDF) for holding and transmitting electronic forms of what are thought of as printed documents. The only representation information needed by today's computer user is the URL for downloading the Acrobat® Reader™. The representation net for PDF files is basically this single node, detailing how to gain access to the software for rendering the data. In reality, it should be an array of nodes with elements for different platforms. All preserved PDF files would reference this one piece of representation information. The

recent appearance of the GNU open-source Xpdf [13] would be reflected by adding it to this array.

Example 2: IBM SCRIPT files

One upon a time, the representation information for a preserved IBM SCRIPT file would point to the IBM SCRIPT program for the IBM/360 platform. Unfortunately we did not have the OAIS model in the 1970s, but if we had had an OAIS archive for storage of our VM/CMS data, this is the only representation information that would have been needed. (Actually the CMS file-type of SCRIPT performed the rôle of representation information, much as file extensions do today on a PC.)

As the 30+ years elapsed, our putative OAIS archive would have expanded the representation information for SCRIPT by information suitable for more current platforms — including the human readable documentation for a live-ware platform. There would probably also be reference to the Hercules project [14] which allows emulation of IBM/360/370 systems of yesteryear. This need to keep up-to-date was highlighted in the InterPARES project [15].

Example 3: The BBC Domesday Project

In 1986, to commemorate the 900th anniversary of the Domesday Book, the BBC ran a project to collect a picture of Britain in 1986, to do so using modern technology, and to preserve the information so as to withstand the ravages of time. This was done using a micro computer coupled to a Philips LaserVision player, with the data stored on two 12" video disks. Software was included with the package, some on ROM an some held on the disks, which then gave an interactive interface to this data. The disks themselves are robust enough to last a long time, but the device to read them is much more fragile, and has long since been superseded as a commercial product.

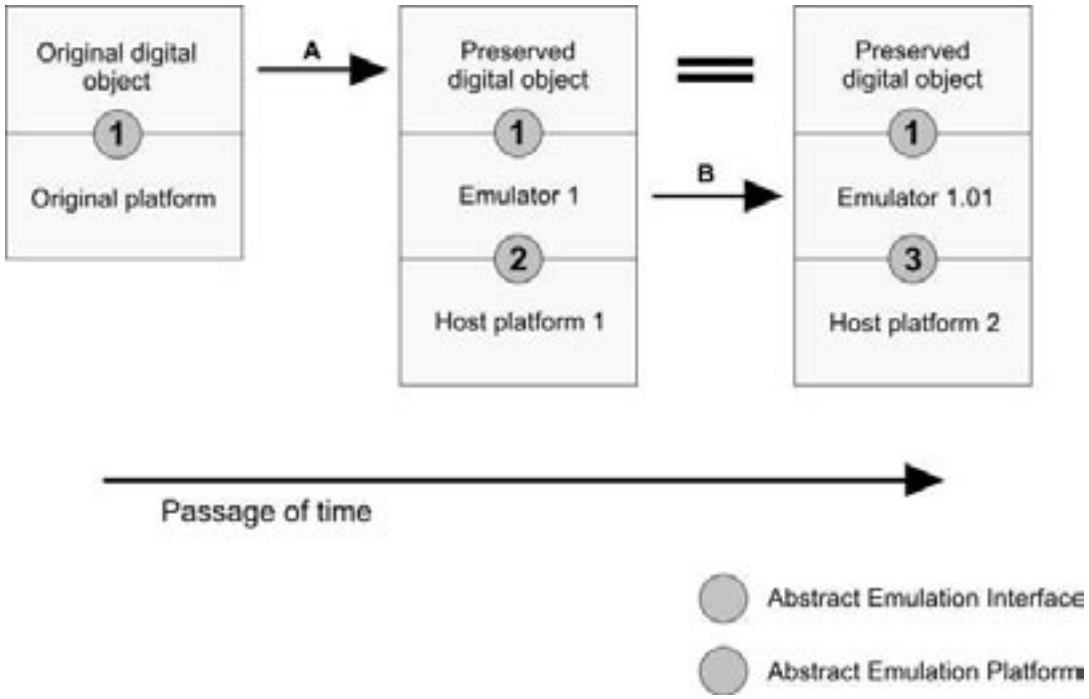
Here we have a clear example where the preservation decisions placed (mis-placed) faith in the media technology of the day, and more crucially in the survival of the information technology practices of the time.

The CAMiLEON project used this example as a test case to show the effectiveness of emulation as a preservation technique. A detailed treatment is to be found on the CAMiLEON web site [16].

We can look at this example with particular reference to its long-term viability, both with regard to the original efforts in 1986, and to the emulation work of 2002. We shall use it to illustrate our ideas about the appropriateness of emulation software as part of the representation information.

Firstly, a bit of background to the work.

We have taken our own advice and preserved the data from the original disks as abstract byte-streams. We can represent this step as the process marked A in the diagram (taken from reference [7]):



The technique was to show that we could use emulation to bridge from the Original platform to a different host platform, labelled Host platform 1 in the diagram. The ingest step (marked A in the diagram) involves identifying the significant properties of the original. The data consisted of the four disk surfaces, each with 3330 tracks, and some software in ROM held inside the BBC micro computer. Some tracks were video images and some held digital data which was often textual. We preserved the ROM contents straightforwardly as binary files, and made each track of the disk into a binary file of pixels for the video images, and a straightforward binary file for each of the digital data tracks. This we claim preserves the significant properties of the software and data necessary for it to run on the BBC computer with its attached video disk player. An example representation network describing the capture process was constructed as part of the Representation and Rendering Project [17]

To demonstrate the validity of this claim, we produced the emulator shown as Emulator 1 on the diagram. The original software relied on an order code and an API (applications program interface) labelled 1 in the diagram. In order to achieve successful preservation of this digital object, we need to reproduce this API with software that operates with a more modern API, labelled 2 in the diagram.

The emulation of the BBC micro-computer was obtained from an open-source emulation written by an enthusiast (Richard Gellman) and available on the net [18]. Although the achievements of enthusiasts are not always ideally structured for use in digital

preservation work, they can often provide a useful starting point for further development. At the very least the source code can act as a handy reference point for new work.

The emulation of the video disk player was done by our own project staff. This emulation software then becomes the major component of the representation information for this data. Its longevity depends crucially on the longevity of the interface labelled 2. Here we have used code that is written in C, and makes use of only a few Win32-specific API calls. In other words our interface labelled 2, is not the whole API of Host platform 1, but only the facilities that we have chosen to use. The move to another platform is made easier by choosing to use as few as possible of the proprietary features of Host platform 1. We may need to recode a few bits of the screen driving routines, but by and large we can expect to find on Host platform 2 an API (shown as 3) that has most of the features needed on the new platform. We expect that a slightly revised emulator called Emulator 1.01 will readily be generated (step B) to run on Host platform 2. Meanwhile, the preserved digital object will be completely unchanged, as indicated by the large equals sign.

Example 3: The BBC Domesday Project — Evolution of Representation Information

At the outset, the storage media consisted of two 12" video disks. The representation information (a booklet supplied with the disks) basically said buy the appropriate hardware including the two E-PROM chips holding software that is used in accessing the video disk player. In addition, the BBC microcomputer had a well documented API for applications programs. This API (or preferably the subset of this that happened to be used) provides the interface labelled 1 in the diagram.

Our preservation of the data from its original preservation medium created byte-streams that closely mirrored the actual physical data addressing. This maximised the validity of the existing representation information, *viz.* the documentation of the API mentioned above.

The emulator then implements this API, opening up the question of the API upon which it itself runs. Thus we add to the representation information the emulator, and the information concerning the API needed to run it. This is not yet stored in a real OAIS archive, but we do have the materials necessary to achieve this, and data from the disks is stored in our LEEDS archive[19].

Our care in producing an emulation system that is not tied too closely to the platform upon which it runs illustrates our desire to produce representation information that will indeed stand the test of time by being easily revised to accommodate newly emerging technologies. This revised emulator becomes an addition to the representation information, extending the easy availability of the original data to a new platform. InterPARES [15] identified clearly the desire of users to access the material on the technology of their own time.

So why emulate in this case? The interactive nature of the digital object is really a part of it. There is no readily available current product that reproduces that interaction, so we treat the interaction software as part of the data to be preserved. On the better examples of current desk-top hardware, it runs faster than the original.

Share and Cross-Reference Representation Nets

We have argued earlier for the impossibility of producing an adequate standard for representation information which will retain its relevance over the decades. To attempt to do so would stifle research and development. We must therefore expect that different data storage organisations may develop different forms of Representation Information. Initiatives such as the PRONOM [20] file format database and the proposed Global File Format Registry will also produce valuable resources that should be linked from representation information.

It would seem that collaboration should be the watchword here.

The emerging solutions for IBM SCRIPT files in example 2 are likely to be applicable to any institution holding such data. With our proposed global namespace, they can all reference the same representation net, and benefit from advancing knowledge on the rendering of such files.

Global Considerations

The implementation of preservation on a global basis means that there will be no overall command. Co-operation will have to be by agreement rather than by diktat. This situation has some aspects that resemble the problems of achieving true long-term preservation. We cannot predict the future accurately, nor can we control it to any great extent, so the ambition to operate on a global scale despite being unable to control activities everywhere in the world sits well with the need for future-proofing. The future is another country whose customs and practices we cannot know.

Referential Integrity

We are proposing that no object that has a name in the digital store is ever deleted. It may be modified, but never deleted. Thus, anyone may use a reference to an object in the OAIS digital storage world confident that it will never become a dangling pointer.

However, the representation information in any OAIS archive will need to refer to information outside its control. (This is actually an inevitable consequence of Gödel's incompleteness theorem — reflected in Cedars by describing nodes holding such references as Gödel ends.) Many of these external references will relate to the current practice of the time.

A vital part of the management of such an archive will involve keeping an inventory of all such external references, and maintaining a process of review of the inventory in the

search for things that are no longer generally understood or refer to information that is no longer available. The remedy in such cases is to update the referring nodes to reflect the new realities. Clearly it is in the interests of good management to try to keep such nodes to a minimum.

For example, a store would have a single node that describes the current version of Microsoft Word to which the representation information for any ingested Word file would refer. When this version becomes obsolete, this one node is updated with information on how to access data in the old format, or to convert to a newer format.

The two level naming proposed earlier helps greatly in implementation of such a policy.

Digital Curation in the UK

The education funding authorities in Britain are currently in the process of setting up a digital curation centre [4]. This is seen as a centre for oversight and co-ordination of digital storage, and for R&D. The decision was announced shortly before Christmas. The centre will be based in Edinburgh, the home of the existing e-Science Centre [21], and EDINA [22].

The centre will not be a repository for the data itself.

It will provide consultancy and advice services, and a directory of standard file formats.

There will be a significant research activity, and a particular focus on digital integration, the enabling of research combining data from different sources.

Academia is addressing its own problems, but what about the rest of the world of digital information, e.g. engineering data? How confident are we that the CAD data for nuclear power stations has an appropriate lifetime, or even half-life?

Summary

We argue strongly for retention of the original in the form of a byte-stream derived as simply as possible from the original data, and for the use of representation information to enable continued access to the intellectual content.

We take the view that for much material it is impossible to have perfect representation information at the time of ingest, but that we must preserve the data and develop its representation information over time.

Ideas on the nature of representation information will evolve over time. We must have systems capable of taking on board changing schemas of representation information.

A two-level naming system, separating reference ID from location (and translating between them) should be the practice for implementing pointers in an OAIS archive, as a

prerequisite for our proposed policy of evolving representation information over time, and sharing it on a global scale.

A Footnote on Bits versus Bytes

The OAIS model uses the bit as the lowest level. However, the byte is the ubiquitous unit of data storage. In today's systems one cannot see how the bits are packed into bytes. When a file is copied from one medium to another we know that whether we read the original or the copy, we shall see the same sequence of bytes, but we know nothing of the ordering of bits within the byte, and these may be different on the two media types. On some media (e.g. 9-track tape) the bits are stored side-by-side.

Pragmatically, we regard the byte as the indivisible unit of storage. If the OAIS model requires us to use bits, then we shall have a single definition of the assembly of bits into a byte. This would enable us unambiguously to refer to the millionth bit in a file, but not constrain us to hold it immediately before the million-and-oneth bit.

References:

[1] Cedars project http://www.leeds.ac.uk/cedars/
[2] CAMiLEON project http://www.si.umich.edu/CAMILEON/
[3] Representation and Rendering Project http://www.leeds.ac.uk/reprend/
[4] UK National Digital Curation Centre
http://www.jisc.ac.uk/index.cfm?name=funding_digcentre
[5] Reference Model for an Open Archival Information System (OAIS) ISO 14721:2002:
http://www.ccsds.org/documents/pdf/CCSDS-650.0-B-1.pdf
[6] The Medium is NOT the message. *Fifth NASA Goddard Conference on Mass Storage Systems and Technologies* NASA publication 3340, September 1996. http://esdis-it.gsfc.nasa.gov/MSST/conf1996/A6_07Holdsworth.html
[7] Emulation, Preservation and Abstraction, RLG DigiNews vol5 no4, 2001.
http://www.rlg.org/preserv/diginews/diginews5-4.html#feature2
[8] Research and Advances Technology for Digital Technology : 6th European Conference, ECDL 2002
http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2458&spage=516
[9] *Migration - A CAMiLEON discussion paper —Paul Wheatley, Ariadne, Issue 29(September 2001)* http://www.ariadne.ac.uk/issue29/camileon/
[10] Cedars architecture. http://www.leeds.ac.uk/cedars/archive/architecture.html
[11] NISO technical metadata standard for still images.
http://www.niso.org/committees/committee_au.html
[12] *C-ing Ahead for Digital Longevity*
http://www.leeds.ac.uk/CAMiLEON/dh/cingahd.html
[13] Xpdf Acrobat® renderer http://www.foolabs.com/xpdf/about.html
[14] Hercules IBM Emulator http://www.schaefernet.de/hercules/index.html
[15] InterPARES project http://www.interpares.org/book/index.cfm, see also [23
[16] Domesday. http://www.si.umich.edu/CAMILEON/domesday/domesday.html

[17] Representation and Rendering Project case study
http://www.leeds.ac.uk/reprend/repnet/casestudy.html
[18] Richard Gellman and David Gilbert, BBC Emulator
http://www.mikebuk.dsl.pipex.com/beebem/
[19] LEEDS archive http://www.leeds.ac.uk/iss/systems/archive/
[20] PRONOM http://www.records.pro.gov.uk/pronom/
[21] EDINA http://www.edina.ac.uk/
[22] UK National e-Science Centre http://www.nesc.ac.uk/
[23] InterPARES2 http://www.interpares.org/ip2.htm

FIBRE CHANNEL and IP SAN INTEGRATION

Henry Yang
McDATA Corporation
4 McDATA Parkway, Broomfield, CO 80021,
Tel: +1-720-558-4418
e-mail: Henry.yang@McDATA.com

Abstract
The maturity and mission-critical deployment of Fibre Channel (FC) in storage area networks (SANs) creates a unique class of multi-terabit networks with demanding throughput, latency, scalability, robustness, and availability requirements. This paper reviews the state of and critical system-level requirements for SANs. It describes how Internet SCSI (iSCSI), FC over IP (FCIP), and Internet FC Protocol (iFCP) integrate with FC SANs and discusses associated benefits and challenges. Finally, the paper examines case studies in performance and protocol tuning in high-speed, long-delay networks, which are increasingly critical for FC-to-IP integration opportunities and challenges.

1.0 Introduction

Information technology (IT) is a key driver and challenge for businesses, government, and research/development centers. Data centers are a critical asset and provide the infrastructure that houses information processing, storage, and communication resources. Corporations are under tremendous pressure to manage return on investment, massive growth in information processing and storage needs at a global scale, management, performance, availability, and scalability requirements, and the IT infrastructure. To add to the challenges, there are many new technology and deployment decisions that have significant implications in terms of value and impact to the data center.

SANs are a critical part of the data center, and are based on high speed, high bandwidth, low latency, and low error rate interconnects for scaling application, database, file, and storage services. FC is the key technology and standard that drive rapid growth of SAN deployment. The development of global and distributed file systems, content-addressable storage, object-oriented storage, cluster and blade servers, and utility computing is driving more integrated IP and FC network usage. The evolution of the data center and new information and computing trends drives the data center toward a more dynamic resource and performance provisioning and management model, which demands more efficient and scalable computing, information storage, and networking. In addition, business and operational requirements in the data center drive the scaling and evolution of larger SANs encompassing metropolitan and wide-area distances, high security and availability, and multi-protocol networks. In the face of these trends, ease of use, configuration, and management of the SAN is even more important.

This paper reviews important requirements and deployment examples. It describes emerging IP SAN technologies and how these technologies interface and integrate with

FC. It also examines several protocol and design considerations, system-level behaviors, and areas that need further research and enhancement. This paper leverages the efforts of many engineers, architects, and researchers from the industry. The paper uses their findings and recommendations, and tries to relate them to SAN applications.

2.0 The FC SAN Today

2.1 FC SAN Overview

FC technology [1] and product deployment has evolved from 1 gigabit per second (Gbps) to 2 Gbps links, and there is development to introduce 4 Gbps and 10 Gbps links. An FC network or fabric is a multi-terabit, low-latency switching network, mainly used to interconnect servers to storage. Although a FC fabric is designed to support any-to-any connectivity, the actual use tends to be some-to-some. Each server talks to a few storage devices or each storage device talks to a few servers, with occasional traffic for backup or other purposes involving devices shared by many sets of storage and servers. Deployment of mid-range to high-end FC fabrics is based on FC directors [2], which are high-availability switches with high-aggregate switching bandwidth and high port density. For the edge part of a large or small fabric, smaller and lower-cost FC switches are typically used. Directors and switches use one or more interswitch links (ISLs) to connect and form a larger fabric. It is common to deploy one or more isolated FC fabrics, called SAN islands. SANs are also extended to campus, metropolitan, and wide-area distances using T1/T3, ATM, IP, SONET, dark fiber, and DWDM technologies.

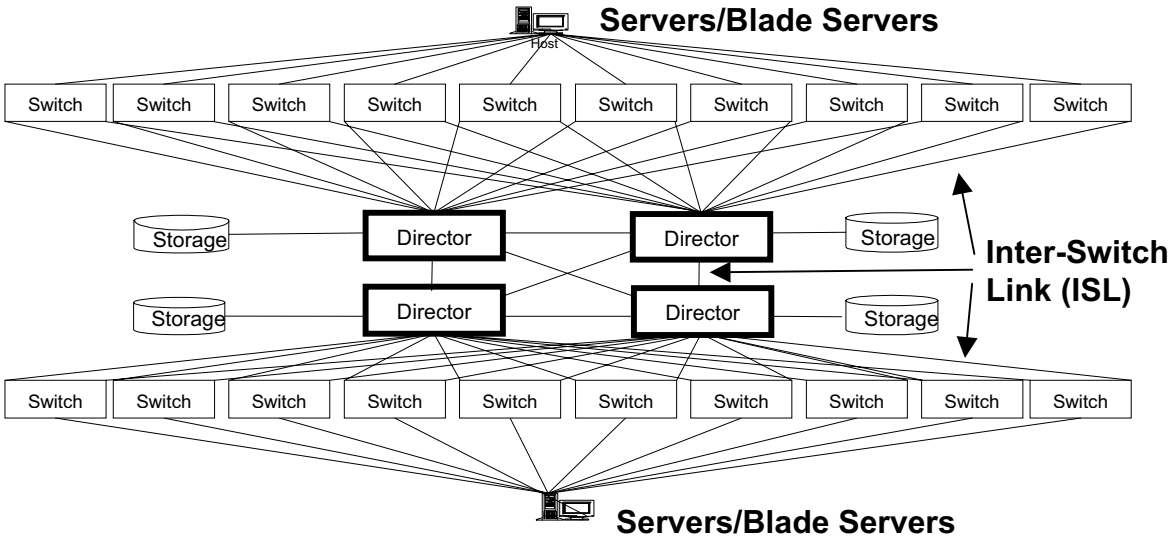


Figure 1 Example of a Large FC Fabric

Figure 1 shows an example of a large (approximately 1000 node) fabric, with directors and switches configured to provide high availability and high-aggregate bandwidth. Servers are typically aggregated at the edge of the fabric, and storage arrays are typically configured near the core of the fabric. It is typical to over-subscribe server link bandwidth in comparison to storage array link bandwidth (more servers with respect to a given storage array). A network of directors forms the core (or backbone) of the fabric.

For a fabric to be operational, there is a fabric initialization, involving all switches, directors, and devices. Initialization steps include parameter exchanges, Principal Switch selection, address assignment, path computation, and zone merge operations. As part of the path computation, directors and switches in a fabric run Fabric Shortest Path First (FSPF) routing protocol to build the forwarding data base. FSPF is a link state protocol that computes shortest path routes for frame forwarding. Within a FC fabric, there are name services and state change notification protocol services for resource discovery, configuration, and change management. FC zoning is an overlay network mechanism to limit the visibility and connectivity of servers to storage devices. A device can be in one or more zones, thereby enabling the sharing of servers (or clusters of servers) and storage resources. When an ISL changes state, all these protocols normally run, and when an end device comes up or goes down, name services and state change notification services run. These services consume more and more resources as the fabric size grows.

2.2 Traffic Patterns in Fibre Channel

Most FC traffic uses the SCSI-FCP protocol [4] on top of FC Class 3 (unacknowledged datagram) service. SCSI-FCP is a request-response protocol that provides frame sequencing within transactions provided by lower-layer FC protocols. On frame loss or error, the protocol performs a transaction-level time out and retransmission. No retransmission of individual frames is supported. Time-out values are typically pre-configured and not based on actual round trip delay. The performance of SCSI-FCP is therefore sensitive to frame loss or frame level errors. Table 1 shows example read and write transactions and protocols frames.

Transaction	**Protocol Direction**	**Frame Type**	**Typical Frame Length**
Read	*Server to Storage*	*FCP_CMD (Read)*	*68 Bytes*
	Storage to Server	*FCP_XFER_RDY*	*48 Bytes*
	Storage to Server	*FCP_DATA (one or more)*	*Up to 2084 Bytes*
	Storage to Server	*FCP_RSP*	*64 Bytes*
Write	Server to Storage	FCP_CMD (Write)	68 Bytes
	Storage to Server	FCP_XFER_RDY	48 Bytes
	Server to Storage	FCP_DATA (one or more)	Up to 2084 Bytes
	Storage to Server	FCP_RSP	64 Bytes

Table 1 Example SCSI-FCP Read and Write Protocol Frames

As bandwidth and delay product increases, it is critical to understand performance tuning and error recovery mechanisms. For configurations with long delay, it is important to consider the way data is moved (write or read). As shown in Table 1, the write transaction has one additional round trip delay more than the read transaction. Therefore, the read operation is faster when network delay is long.

2.3 Critical Factors in SAN Deployment

SAN deployments today range from small fabrics with less than 100 devices to large fabrics with several thousand devices. The following are factors critical to SAN design and deployment:

- High availability: The impact of down-time and lost of information to business is severe. High availability requirements are quantified to vary from several 9's, to 99.999%, to no down time. Most highly available fabrics are based on dual-rail redundancy and highly available directors, switches, and gateways. Servers and storage devices may have redundant paths through one fabric or through separate redundant fabrics with no shared single point of failure. Directors and some switches are designed with high-availability features, including fully redundant and hot swappable field-replaceable units (FRUs) and hot software download and activation, meaning that operation may continue through a software upgrade.

- Robustness and stability: Some FC servers, associated host bus adapters (HBAs) and storage devices are extremely sensitive to frame loss and frame out of order delivery. Error recovery in the SCSI-FCP protocol is based on command and transaction level time-out and retry. Therefore, SCSI-FCP expects very low frame loss rate, since frame loss has significant performance impact. The design of SANs has to account for the following factors:

    - It is important to limit and reduce FC fabric size in terms of number of switching nodes. The goal is to limit the frequency of fabric initialization, FSPF route computation, and traffic for state notification and name services.

    - It is critical to ensure there is adequate aggregate bandwidth (fabric-wide and for individual links), to avoid severe and prolonged congestion. FC fabrics use a link-level, credit-based flow control, which is useful for handling short-term, bursty congestion. In FC, it is not common to use active queue management techniques (e.g., based on random early detection) to minimize queue build up. It is typical for a FC switch to discard frames that have been queued for a pre-determined time (e.g., 0.5 to 1.0 second), as part of the stale frame discard policy. As the deployment of multi-speed (1 Gbps, 2 Gbps, 4 Gbps, and 10 Gbps) ramps up, the design of the network and switching architecture becomes more challenging. As the size of network grows, comprehensive congestion management mechanisms become more critical and current link-level flow control may no longer be adequate.

- Performance: Most FC switches and directors specify best-case frame latency to be less than a few microseconds. But latency grows with loading and can result in effective bandwidth to be significantly less than nominal bandwidth. Measured frame latency at 70% link utilization [3] showed it was 5.2 to 6.5 microseconds for one vendor's product and 2.6 to 2222.6 microseconds for another vendor's

product. The lesson is that not all switches are designed equal. Switching architecture issues like head-of-line blocking and internal resource bandwidth (throughput or frame rate) limitations impact throughput, latency, and congestion loss, especially at higher offered load.

- Distance extension: Requirements for disaster recovery and business continuance (file/data mirroring, replication, and backup) are driving the deployment of SAN extension to deliver better performance and availability. In addition to robustness, stability, and performance considerations, it is important to understand the configurations, products, and protocols and system tuning parameters with respect to distance extension technology. We examine this topic later.

- Scaling the SAN: A large number of FC fabrics deployed today are small islands of fabrics that are not inter-networked into a large and connected SAN. Reasons for deploying isolated islands include early adopters learning new technology, difficulty and lack of confidence in management and operational stability of a large fabric, and insufficient business and operational drivers (for connecting islands of FC fabrics). However, there are many benefits of internetworking FC islands. Resource sharing (such as tape library for backup) and the ability to dynamically provision and allocate resource are some of the benefits. When scaling an FC SAN, it is important to maintain performance and availability properties. Since a FC fabric is similar to an IP layer 2 switching network, it is important to constrain the number of switches in a fabric so the resulting fabric is stable and robust. When interconnecting FC fabrics, it is critical to consider isolating FC fabric local initialization and services, while allowing servers and storage devices to be interconnected regardless of locality. This is an area of further research and standardization work, and currently ANSI T11 has a fabric extension study group addressing these topics.

3.0 FC & IP Integration & Challenges

3.1 IP SAN Developments

The emergence of iSCSI, FCIP, and iFCP standards [5, 6, 7, 8, 9] enables IP technology to enhance the deployment and benefits of SANs. FCIP and iFCP protocols use a common framing and encapsulation design. We examine the applicability, design, and limitations of these technologies in the following sections. These protocols leverage the matured IPSec standard and technology to enable security (including authentication, integrity, and privacy). As part of the protocol suite, Internet Storage Name Service (iSNS) [10] provides a method to manage and configure names, registry, discovery, and zones for multi-protocol SANs. The use of Service Location Protocols (SLP) [11] to discover services and resources is another critical part of the standard.

3.2 iSCSI

iSCSI is a SCSI over TCP transport protocol used between a SCSI initiator and a SCSI target for storage-block level transport of SCSI commands and payloads. iSCSI protocol uses TCP/IP and IPSec as its network transport and security protocols. It has many features designed to leverage standard TCP/IP protocols to block storage needs. These features include the use of multiple TCP connections (for a given session), cyclic redundancy check (CRC) digests, out of order data placement, and TCP connection failure recovery options. iSCSI design and analysis have been presented in several papers [12, 13, 14, 15, 16].

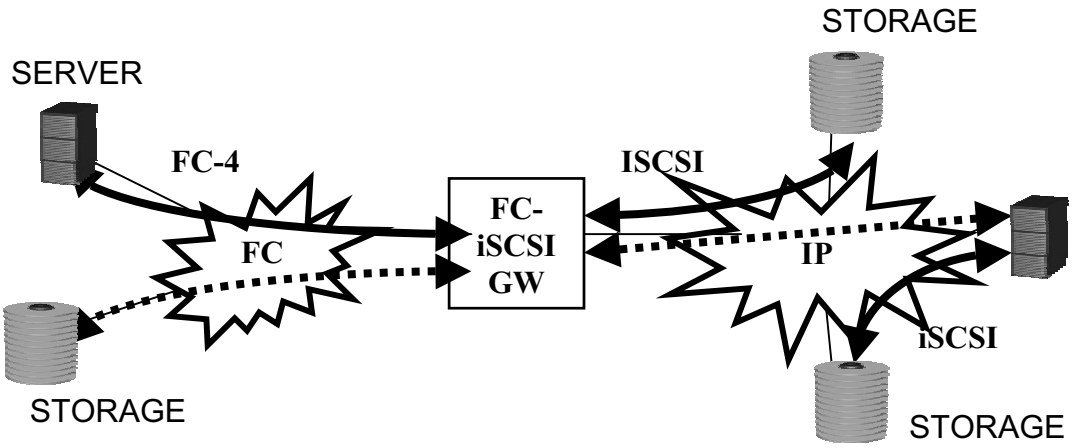


Figure 2 FC-iSCSI Gateway

In Figure 2, the FC-iSCSI gateway provides the internetworking of iSCSI devices with FC devices, while communicating with each of the networks appropriately. While the FC SAN and the IP SAN are operating independently, the gateway maps selected iSCSI devices into the FC SAN and selected FC devices into the IP SAN. When a FC server creates a SCSI-FCP session to a storage device, the gateway intercepts the request and acts as a proxy for the storage device. On the IP side, the gateway acts as a proxy initiator (for the server), and creates an iSCSI session for the storage device. The gateway maintains and manages the state of the gateway portion of supported sessions. For an IP-based server creating an iSCSI session to a FC storage device, the gateway performs similar roles as proxy target on iSCSI session and proxy initiator for the SCSI-FCP session.

An iSCSI gateway performs several important functions, including FCP and iSCSI session-level protocol translations, command and payload forwarding, error checking, and command/session-level error propagation. A gateway has to manage device discovery and registry (on the IP side with an iSNS server, and on the FC side with FC name services), authentication of FC and IP devices, and mapping of device names to local addresses, etc. It is important that a gateway is as transparent as possible to the servers and storage devices using the gateway, while maintaining high data integrity. It

should have very low latency and sufficient bandwidth to forward commands and payloads, and support a sufficiently large number of sessions to enable storage consolidation (a high end storage array on the FC side shared by a large number of IP based servers). Management of the multi-protocol SAN is a critical part of the deployment success.

3.3 FCIP

FCIP is a tunneling protocol that transports all FC ISL traffic. Similarly, FCIP uses TCP/IP as the transport protocol and IPSec for security. A FCIP link tunnels all ISL traffic between a pair of FC switches, and may have one or more TCP connections between a pair of IP nodes for the tunnel end points. From the FC fabric view, an FCIP link is an ISL transporting all FC control and data frames between switches, with the IP network and protocols invisible. One can configure one or more ISLs (using FCIP links) between FC switches using FCIP links. Figure 3 shows an example of FCIP links being used as ISLs between FC switches A and B.

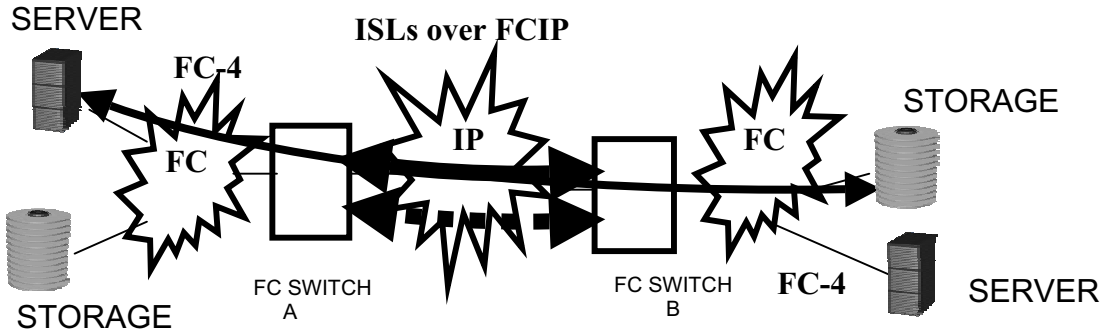


Figure 3 FC-FCIP Tunnel

A key advantage of the FCIP tunnel approach is transparency to a fabric, as existing fabric tools and services are used. Once a FCIP link is configured, existing fabric operations and management continue. Similarly, fabric initialization, FSPF routing protocol, and name/state change services run transparently over FCIP links. However, since FC fabric-level control protocols run over the FCIP tunnel, IP and TCP connection failures can disrupt the FC fabrics on both sides. Given the speed and bandwidth differences between FC and a typical IP network used to interconnect remote SANs, the design and management of congestion and over-load conditions is important to understand.

For the FCIP tunnel, a simple FIFO (first in first out) frame forwarding queue design can result in head-of-line blocking of fabric initialization protocol frames when the tunnel is congested, or the TCP connection is in slow-start recovery mode. Another case to consider is when a SCSI-FCP transaction time out occurs, the entire transaction (such as 1 MB block) might be retransmitted over an FCIP link that is experiencing congestion. In addition, there might be multiple application streams using the same FCIP link, and there is no mechanism to help reduce or avoid network congestion. These are possible

scenarios of overload and congestion that can result in performance and stability issues that impact the entire fabric. For a medium to large fabric, these are critical issues for concern. Most FCIP deployments are based on small fabrics, where there are a small number of devices and switches at each end of the FCIP link, and these issues are less critical.

3.4 iFCP

iFCP technology is a gateway-to-gateway protocol for providing FC device-to-FC device communication over TCP/IP. For each pair of FC devices, there is an iFCP session created between a pair of gateways supporting the devices. An iFCP session uses a TCP connection for transport and IPSec for security, and manages FC frame transport, data integrity, address translation, and session management for a pair of FC devices. Since an iFCP gateway handles the communications between a pair of FC devices, it only transports device-to-device frames over the session, and, hence, the FC fabrics across the session are fully isolated and independent. This is a major difference between iFCP and FCIP, in that FCIP builds an extended fabric, tunneled over IP.

In contrast to an FC-iSCSI gateway, an iFCP gateway transports FC device-to-device frames over TCP, and in most cases original FC frames, including the original CRC and frame delimiters, are transported. An FC-iSCSI gateway terminates and translates SCSI-FCP protocol from the FC side and similarly terminates and translates iSCSI protocol from the IP side.

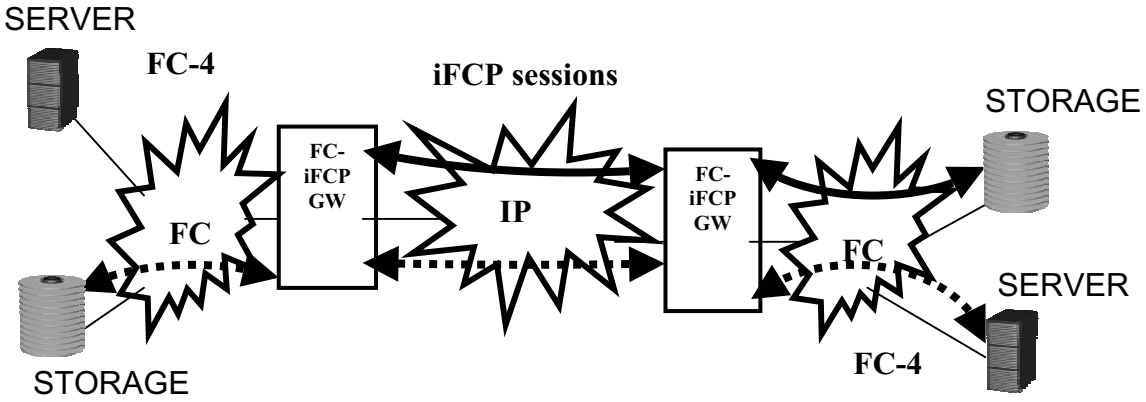


Figure 4 FC-iFCP Gateway

The iFCP draft standard specifies an address-translation mode as well as an address-transparent mode, depending on whether the FC addresses of devices are translated or not. An FC device exchanges login and protocol parameters with another FC device using FC link service protocol frames as part of the session creation and parameter exchange protocol. In address-translation mode, a gateway intercepts these device-to-device link service protocol frames and translates device addresses embedded in the frames. It regenerates frame CRCs when the original frame content is changed, which imposes extra overhead on the iFCP gateway. Address translation is a particularly useful feature when interconnecting FC fabrics. It enables installation of a gateway between existing

fabrics without requiring fabric address changes. A gateway manages the session state, addresses translation and mapping, and provides proxy functions for remote devices. In addition, a gateway performs security functions (like authentication of devices), and works with an iSNS server for registry and discovery functions.

The configuration and management of an iFCP gateway is more involved than for an FCIP gateway, as each device-device session has to be set up. Also, an iFCP gateway has more device proxy-related states to manage. As the number of device-to-device sessions increases, an iFCP gateway design becomes more complex and may result in performance and stability issues. However, one can use admission control techniques to limit the number iFCP sessions allowed for a gateway. Since an iFCP gateway is managing device-to-device communications, it can enforce some degree of flow control by pacing command forwarding at the time of congestion. The iFCP specification allows an optional unbounded connection feature, which sets up and uses a pool of backup TCP connections for fast-session fail-over support. This assists a gateway in providing faster connection fail-over.

3.5 TCP/IP & Transport Protocol Discussions

Some classes of applications have different requirements for transport services and protocols. For example, applications that prefer timeliness in delivery over reliable data delivery (such as RealAudio, Voice over IP) prefer a different transport service and protocol design [17] than that of TCP. Also, for applications that prefer a different type of fault tolerance, reliability, and a non-byte stream-oriented transport service, a different type of transport protocol might be needed (such as Stream Control Transmission Protocol (SCTP) [18]). These are examples of new transport protocol research and standard development activities. TCP protocol is undergoing many enhancements to improve performance under different operating conditions, and these enhancements include High Performance Extensions (TCP Window Scaling Option, Round-Trip Time Measurements, Protect Against Wrapped Sequence Numbers) [19], Selective Ack Option [20, 21], Explicit Congestion Notification [22, 23], Eifel Detection Algorithm [24], and High Speed TCP (HSTCP) [25].

As part of the design considerations for an IP SAN, the design and tuning of TCP for the SAN is critical. For iSCSI servers and storage devices, the design and tuning of protocol off-load, zero-copy, interrupt coalescing, and buffer-MTU-MSS tuning are critical (MTU is the maximum transfer unit, MSS is the maximum segment size). For iSCSI, FCIP, and iFCP gateway design, buffer-MTU-MSS tuning is very critical and several of the aforementioned TCP enhancements are important considerations for scaling the IP SAN for 1 to 10 Gbps speeds. For long and fast network (LFN), HSTCP enhancement is an important design. Multiple TCP connections for iSCSI, FCIP link, and unbounded iFCP connections are critical considerations for load balancing and high availability.

In addition to the IP based transport, there are developments for operating Gigabit Ethernet and FC protocol directly over SONET-based transports for Generic Frame Protocol ITU-T G.7041 standards [26].

4.0 Some Case Studies

4.1 Experiment of 10 Gbps Transcontinental Data Access

As part of the Supercomputing Conference 2002 demonstration of SAN extension over a multi-gigabit transcontinental network, [27] test results of an FC SAN interconnected with iFCP gateways over a 10 Gbps link from San Diego to Baltimore were presented. Figure 5 shows the configurations used for the experiment.

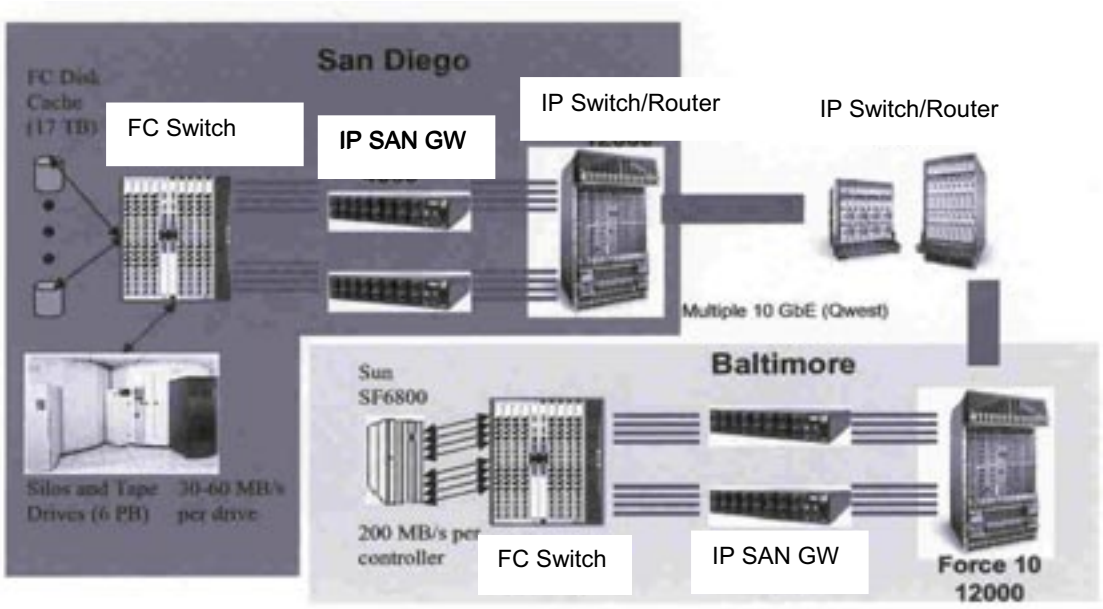


Figure 5 Schematic of Data Access for the SC'02 demonstration

The Supercomputing ’02 experiment proves the operation of a network running FC over IP network, using iFCP gateways, between the San Diego Supercomputer Center (SDSC) and the SDSC booth in Baltimore. The experiment demonstrates that FC traffic, using iFCP gateways, runs over a 10 Gbps link in excess of 2,600 miles, with a round-trip latency of 70 to 90 milliseconds. Aggregate throughput was relatively constant at 717 MB/s, and read performance was slightly better than write performance. In addition to the IP/iFCP based demo [27], there was another experiment of FC traffic over FCIP using a 10 Gbps SONET link, configured between the Pittsburgh Supercomputing Center (PSC) booth and the SDSC booth at the SC’02 show.

4.2 Remote Mirroring

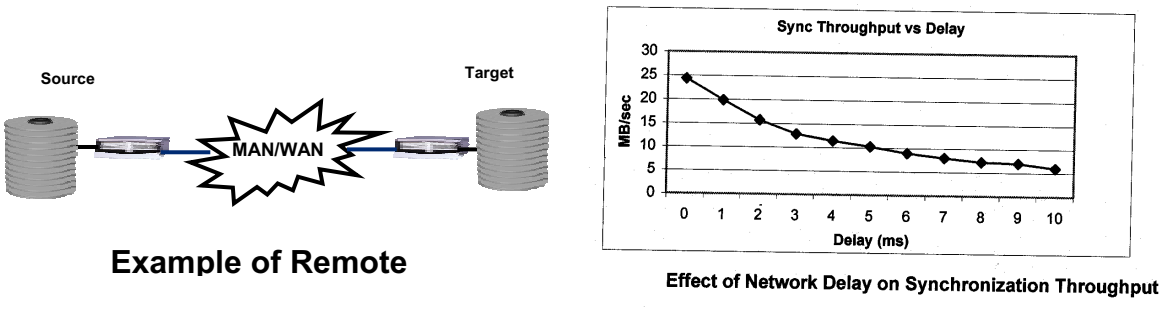


Figure 6 Remote Mirroring – Throughput vs Delay

When performing remote mirroring of logical units (LUNs), remote copy operations must synchronize data copied to each of the LUNs to ensure data coherency within the mirror group. The effective throughput of the remote mirroring of 12 LUNs was shown to drop from 25 MBps to about 5 MBps as the round trip delay increases from 0 to 10 ms, as shown in Figure 6 [28]. It is important to configure and tune file and block size, MTU, MSS, and synchronization rate. In addition, the use of compression to reduce the amount of data transfer is important.

4.3 Delay and Cache Effect on I/O Workload

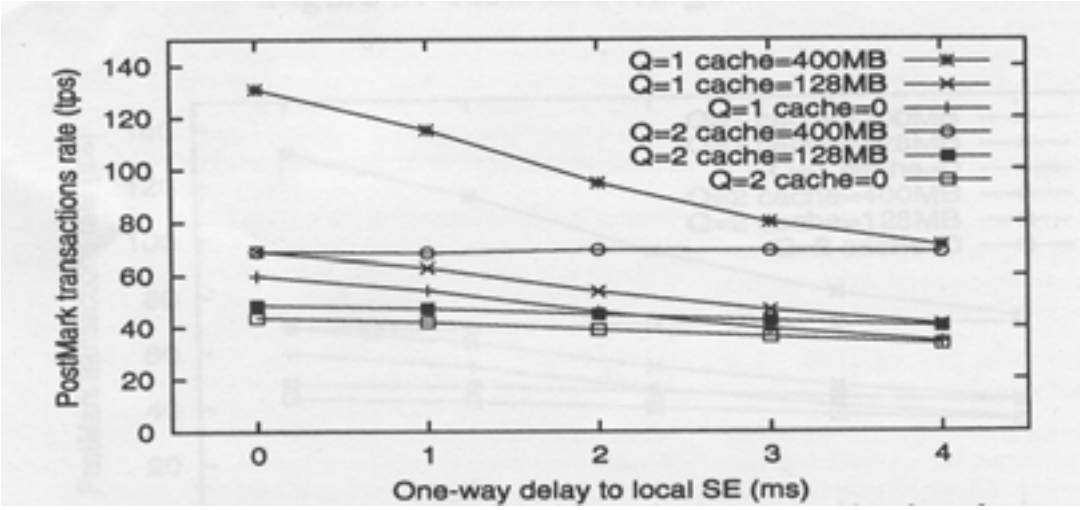


Figure 7 Delay and Cache Effect of PostMark Experiment

PostMark was used to test the delay and cache sensitivity of the I/O workload of a large email server [29]. Figure 7 shows the PostMark transactions rate of I/O from a FreeBSD host to a storage element (SE) with varying delays (to the SE) and cache sizes in FreeBSD VM cache. The transaction rate declines as the delay is increased, and with larger cache sizes the transaction rate increases. Application performance sensitivity with

respect to delay and error recovery is an area that needs further research and understanding.

4.4 Long Fast Network Experiment

In another case [30], the University of Tokyo conducted an experiment using 'iperf' running TCP between Maryland and Tokyo, traversing the Abilene and APAN networks. The result was surprising in that Fast Ethernet is sometimes faster than Gigabit Ethernet on LFN. The main cause of the throughput degradation with Gigabit Ethernet LFN tests was congestion overflow of an intermediate router, resulting in cranking of TCP time out, slow start and congestion control mechanisms. Transmission rate control is important to mitigate the overflow in the bottleneck's buffer in addition to the window size control. Therefore, transmit rate or bandwidth limiting is another important mechanism that avoids or mitigates the impact of congestion overflow in an intermediate network.

4.5 Fast Write

We examine a method to improve the write performance over a long delay network. As shown in Table 1, a SCSI write transaction incurs two round trip delays for a data block. The maximum block size is determined by the target (storage) device's buffer capacity and is specified by the target in the XFR_RDY message. For example, writing one MB of data using 64 KB blocks takes 16 transactions, which is 32 round trips plus data transfer time. Fast Write [31] is a way to minimize round-trip delay overhead and accelerate SCSI write performance leveraging a gateway's buffer capacity. The XFR_RDY is spoofed by the gateway on the initiator (server) side of the network, and the data is buffered by the gateway on the target side of the network until the target sends its own XFR_RDY. In addition, the use of TCP protocol with selective retransmission (on error) provides better frame loss recovery than retransmitting the entire block on timeout (as in the SCSI-FCP case). With Fast Write, the number of round trip involved for a 1 MB transfer is reduced to two round trips.

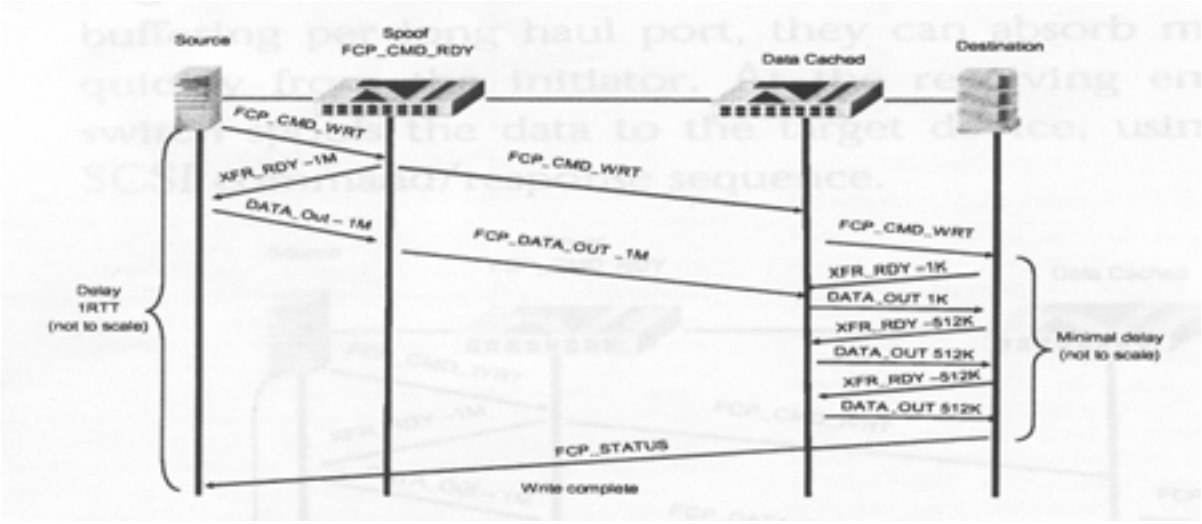


Figure 8 Fast Write Example

Figure 8 shows an example of Fast Write, where a 1 MB transfer is negotiated between the source and the left-hand gateway as well as between the two gateways. For the write operations between the final gateway and destination, the maximum block size is specified by the destination. Most of the round-trips required are over the SAN between the right-hand gateway and destination. Therefore, the SAN should have higher bandwidth, lower latency, and lower error rate than the WAN connection between gateways. Fast Write is an innovative method of using standard protocols to leverage the capability of a gateway and leverage TCP protocol benefits over WAN.

5.0 Summary

We present several of the critical requirements and best practices for FC SAN deployments. We examine IP SAN technologies and protocols, and show that FC and IP integration works well - integrated SANs are a critical part of today's data center. We explore how several new high-speed protocol extensions work, and areas that need further research and development. The deployment of high-speed and long-distance networks for data centers (while providing good performance and reliability) is becoming very important and has potential value as well as challenges.

Acknowledgements

I would like to thank many of the engineers, architects, and researchers who are mentioned in the references and many others who are not explicitly mentioned here. I would like to thank Stuart Soloway for his detailed review and help with this paper. I would also like to thank Tom Clark and Neal Fausset for their review and inputs.

References

1. Latest FC standards and drafts can be found at www.t11.org.
2. H. Yang, "Performance Considerations for Large-Scale SANs", McDATA Corporation, 4 McDATA Parkway, Broomfield, CO 80021, December 2000. www.mcdata.com.
3. Randy Birdsall, "Competitive Performance Validation Test", Miercom Labs, Princeton Junction, NJ, June 2002.
4. Stephen Trevitt, "Traffic Patterns in Fibre Channel", McDATA Corporation, May 2002. Available at www.mcdata.com.
5. RFC 3643 - Fibre Channel (FC) Frame Encapsulation. www.ietf.org.
6. Draft-ietf-ips-iscsi-20.txt, January 19, 2003. www.ietf.org.
7. Draft-ieft-ips-fcovertcpip-12.txt, August 2002. www.ietf.org.
8. Draft-ietf-ips-ifcp-14.txt, December 2002. www.ietf.org.
9. T. Clark, "IP SANs A Guide to iSCSI, iFCP, and FCIP Protocols for Storage Area Networks", Addison-Wesley, ISBN 0-201-75277-8, October 2002.
10. Draft-ietf-ips-isns-21.txt, October 2003. www.ietf.org.
11. Draft-ietf-ips-iscsi-slp-06.txt, December 2003. www.ietf.org.
12. K. Meth, J. Satran, "Design of the iSCSI Protocol", IEEE/NASA MSST2003 Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies, April 7-10 2003, storageconference.org/2003.

13. H. Thompson, C. Tilmes, R. Cavey, B. Fink, P. Lang, B. Kobler, "Considerations and Performance Evaluations of Shared Storage Area Networks At NASA Goddard Space Flight Center", IEEE/NASA MSST2003 Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies, April 7-10 2003, storageconference.org/2003.
14. K. Voruganti, P. Sarkar, "An Analysis of Three Gigabit Networking Protocols for Storage Area Networks", 20th IEEE International Performance, Computing, and Communications Conference, April 2001.
15. S. Aiken, D. Grunwald, A. Pleszkun, J. Willeke, "A Performance Analysis of the iSCSI Protocol", IEEE/NASA MSST2003 Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies, April 7-10 2003, storageconference.org/2003.
16. P. Sarkar, K. Voruganti, "IP Storage: The Challenge Ahead", 10th Goddard Conference on Mass Storage Systems and Technologies/ 19th IEE Symposium on Mass Storage Systems, April 15-18, storageconference.org/2002.
17. Draft-ietf-dccp-problem-00.txt, October 23, 2003. www.ietf.org.
18. RFC 2960 – Stream Control Transmission Protocol, October 2000. www.ietf.org.
19. RFC 1323 – TCP Extension for High Performance. www.ietf.org.
20. RFC 2883 – An Extension to the Selective Acknowledgement (SACK) Option for TCP, July 2000. www.ietf.org.
21. RFC 3517 – A Conservative Selective Acknowledgement (SACK) – based Loss Recovery Algorithm for TCP, April 2003. www.ietf.org.
22. RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP, September 2001. www.ietf.org.
23. S. Floyd, "TCP and Explicit Congestion Notification", Lawrence Berkeley Laboratory, One Cyclotron Road, Berkeley, CA 94704, ACM Computer Communication Review, V. 24 N.5, October 1994, p. 10-23. www.icir.org/floyd/papers.html.
24. RFC 3522 – The Eifel Detection Algorithm for TCP, April 2003. www.ietf.org.
25. RFC 3649 – HighSpeed TCP for Large Congestion Windows, December 2003. www.ietf.org.
26. ITU-T G.7041 Standards, reference www.itu.int/ITU-T/.
27. P. Andrews, T. Sherwin, B. Banister, "A Centralized Access Model for Grid Computing", IEEE/NASA MSST2003 Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies, April 7-10 2003, storageconference.org/2003.
28. EMC Corporation, "MirrorView Using Fibre Channel over IP", December 30, 2002.
29. E. Gabber, J. Fellin, M. Flaster, F. Gu, B. Hillyer, W.T. Ng, B. Ozden, E. Shriver, "StarFish: highly-available block storage", Proceedings of the FREENIX track of the 2003 USENIX Annual Technical Conference, San Antionio, Tx, June 9-14.
30. M. Nakamura, M. Inaba, K. Hiraki, "Fast Ethernet Is Sometimes Faster Than Gigabit Ethernet on LFN – Observations Of Congestion Control Of TCP Streams", University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan, www.data-reservoir.adm.s.u-tokyo.ac.jp/paper/pdcs2003.pdf.

31. McDATA Corporation, “Maximizing Utilization of WAN Links with Nishan Fast Write”, McDATA San Jose Development Center, 3850 North First Street, San Jose, Ca 95134. 2002.

CHALLENGES IN LONG-TERM DATA STEWARDSHIP

Ruth Duerr, Mark A. Parsons, Melinda Marquis, Rudy Dichtl, Teresa Mullins

National Snow and Ice Data Center (NSIDC)
449 UCB, University of Colorado
Boulder, CO 80309-0449
(rduerr, parsonsm, marquism, dichtl, tmullins)@nsidc.org
Telephone: +1 303-(735-0136, 492-2359, 492-2850, 492-5532, 492-4004)
Fax: +1 303-492-2468

1 Introduction

The longevity of many data formats is uncertain at best, and more often is disturbingly brief. Maintenance of backwards compatibility of proprietary formats is frustratingly limited. The physical media that store digital data are ephemeral. Even if the data are properly preserved, the information that allows the data to be searched and which maintains the context of the data is often lost, threatening data utility. These are only a few of the formidable problems that threaten the long-term preservation and long-term use of digital data.

Over the past decade, much has been written about the problems of long-term digital preservation (see for example [14], [15], [32], [38], and [39]). Many approaches or strategies to address these problems have been proposed (see for example [7], [10], and [32]), and a number of prototypes and test beds have been implemented (see for example [44]). No one has developed a comprehensive solution to these problems. In fact, there may not be a single solution.

Most of the literature applies directly to the needs of libraries, museums, and records management organizations. Only rarely are issues related to preservation of science data discussed directly. Stewards of scientific data often face much different issues than the typical library, museum, or records archive. Some issues are simpler others more complex.

In this paper, we provide a brief history of data stewardship, particularly science data stewardship, define long-term stewardship; and discuss some of the problems faced by data managers. We describe a broad array of data stewardship issues, but we will focus on those that are particularly amenable to technological solutions or that are exacerbated when archives are geographically distributed.

2 A Brief History of Scientific Data Stewardship

A cursory review of scientific data stewardship as a discipline distinct from document preservation or records management suggests that it is a fairly recent concept. For most of human history, what little scientific data existed was recorded in notebooks, logs or

maps. With luck, a library or archive would collect and preserve these logs and maps. The archives may have been maintained by the church, a professional society, or perhaps were established through government regulation, but it was generally an ad hoc affair. Unless a potential data user was already aware of the existence and location of certain "data set," it was extremely difficult to find and access the data.

The establishment and growth of academic and public libraries in more recent centuries greatly improved data preservation and access. Libraries were at the forefront of new data cataloging, indexing, and access schemes; librarians were critical data stewards. Yet the "data" were still primarily in the form of monographs and logbooks, and, logically, libraries focused more on books, journals, and other publications more concerned with data analysis. (Maps may have been as readily archived as books and journals). It wasn't until the establishment of the World Data Centers (WDCs) in 1957-1958 that the concept of a publicly funded facility specifically charged with providing data access and preservation became prominent [1].

The World Data Center system originally archived and distributed the data collected during the International Geophysical Year [1]. The data in question were generally small in volume and certainly not digital, but the concept that an institution would focus on the preservation and distribution of raw data as opposed to the interpretation of those data was revolutionary. Furthermore, the WDCs were organized by disciplines such as glaciology and meteorology. This helped reinforce an association between discipline-specific science and data stewardship.

Since then, the number of discipline-specific data centers has grown. In the US a total of nine national data centers were established, primarily sponsored by NOAA, DOE, USGS and NASA [2] to archive and distribute data in disciplines such as space science, seismology, and socioeconomics. The development of these world and national centers made finding relevant data a little simpler. Now there was likely to be an organization that could be queried if only by mail or telephone. If they couldn't provide the data directly, they were usually able to provide references to other places to look.

Local and state governments, universities, and even commercial entities have continued the trend and established a variety of data centers, typically organized around disciplines or subject areas as diverse as "advertising" [3] or "cancer in Texas" [4]. The Federal government again made a significant contribution in the early 1990s when NASA established eight discipline-specific Distributed Active Archive Centers (DAACs) to collaboratively archive and distribute data from NASA's Earth Science Enterprise (ESE).

In some ways the DAAC system followed the model of the distributed and discipline-specific World and National Data Centers, and NASA typically collocated the DAACs with already established data centers [2]. However there are some key differences in the approach. On one hand, DAACs are intended to only archive and distribute data during the most active part of the data life cycle. The DAACs are to transfer their data to a permanent archive several years after each spacecraft mission in the ESE program ends, but the details of this transfer are yet to be finalized. On the other hand, an early and

important goal of the ESE was to make finding and obtaining Earth science data simpler than it had been.

The DAACs are part of a larger system of remote sensing instruments and data systems called the Earth Observing System (EOS). They are linked together through the EOS Data and Information System (EOSDIS) Core System (ECS), which provides tools and hardware to handle ingest, archival, and distribution of the large volumes of data generated by EOS sensors and heritage data sources. An important component of ECS is an electronic interface that allows users to search and access the holdings of all of the DAACs simultaneously. This interface was initially developed as an independent client that users would install on their own machine, but shortly after ECS development started, the first web browsers became available. This led to the development of the EOS Data Gateway (EDG), a web-based search and order tool. Currently the EDG allows search and access to DAAC data as well as data located at several data centers scattered around the world.

What is important to note about ECS and the DAACs is that it was arguably the functional beginning of new model of data management where data archival was geographically distributed, but search and order were centralized. It is also notable that this was a newly comprehensive effort to acquire, archive, and provide access to a very large volume of data but there is still no concrete plan for the long-term disposition of the data. Both these trends—centralized access to decentralized data and inadequate planning for long term archival—continue today. Indeed NASA is moving further away from a data center approach with its new Strategic Evolution of Earth Science Enterprise Data Systems (SEEDS) [12].

Of course, the World Wide Web has been a major driver in the increased decentralization of data storage. Furthermore, improved search engines theoretically make it easier than ever to find data. We have even heard it suggested that Google may be the only search engine needed. General search engines, however, provide little information to help a user determine the actual applicability or utility of the data found. Little of the information currently available on the web has been subject to the levels of peer-review, copyediting, or quality control traditionally done by data managers or library collection specialists [18]. Finally, no mechanism ensures the preservation of much of the information available via the Web. Often web sites cited in a paper are no longer active mere months after the publication of the paper [43].

There are many efforts underway to address some of the issues inherent in distributed Earth-science data systems including the overall Web. Some examples of centralized search tools for distributed scientific data include:

- NASA's Global Change Master Directory (GCMD) (http://gcmd.nasa.gov)
- The Distributed Oceanographic Data System (http://www.unidata.ucar.edu/packages/dods/index.html)

- The Alexandria Digital Library Project (http://alexandria.ucsb.edu/)
- The National Spatial Data Infrastructure (NSDI) (http://www.fgdc.gov/nsdi/nsdi.html)

Some of these efforts predate the World Wide Web, and some like the GCMD are strictly search tools, while others such as the NSDI attempt (with mixed success) to provide actual data access.

As data managers at the National Snow and Ice Data Center (NSIDC), we are primarily concerned with Earth science data, but we should note that many of the issues we will discuss apply to a variety of disciplines. Based on some of our experience at a session on "Virtual Observatories" at the Fall 2003 meeting of the American Geophysical Union, it seems that non-Earth Science related disciplines sometimes lag behind the Earth sciences in the management of their data. Mechanisms for simultaneously searching and accessing data stored at multiple distributed data centers may not exist. For example, no equivalent to the GCMD or EDG currently exists for the solar or space physics community. This situation is rapidly changing. Numerous groups are working on virtual observatory concepts, which in some ways are reminiscent of the EOS DAAC system described earlier.

We should also be aware of the growth of private records management companies. It is certainly possible for commercial entities to address some of the issues of modern data stewardship, but very little research has been done to accurately quantify the necessary costs of a distributed data management infrastructure. Nor have there been any significant efforts to do a cost-benefit analysis of the various components of such a structure [17]. This is especially true in the international context, where not only is distributed data management more challenging, but cost models become more difficult. For example, different countries have data access and pricing policies that are rooted less in economics than in political or philosophical issues such as the right for citizens to access government documents (See [17] and [16] for examples.).

In the following sections, the challenges of providing distributed data discovery and access, while adequately addressing long-term stewardship will be discussed. NSIDC's more than 25-year history as:

- A World Data Center
- Part of a NOAA cooperative institute
- A NASA Distributed Active Archive Center (DAAC),
- NSF's Arctic System Science (ARCSS) Data Coordination Center ADCC) and Antarctic Glaciological Data Center (AGDC)

- A central node for the International Permafrost Association's (IPA) Global Geocryological Data System (GGD)

will serve as one source of examples.

3 Long-Term Stewardship Defined

Within the data management field, "long-term" is typically defined as:

> *A period of time long enough for there to be concern about the impacts of changing technologies, including support for new media and data formats, and of a changing user community, on the information being held in a repository[5].*

Given the current rate of technological change, any data-generating project or program with a duration of five or more years should be considered as long-term and will need to take changes in technology into account.

Stewardship, especially data or scientific stewardship is more difficult to define. Of the 107 results recently found with a Google search of the phrase "scientific stewardship," very few (primarily NOAA sites, a few religious sites, and one lumber company) actually defined what the phrase meant in their context. These concepts are relatively new and do not show up in standard information science dictionaries or encyclopedias.

The term data stewardship was used in the early 1990s by the Department of Defense in DOD Directive 8320.1-M.1, which defined data administration as "the person or group that manages the development, approval, and use of data within a specified functional area, ensuring that it can be used to satisfy data requirements throughout the organization" [40].

Two other relevant definitions can be found in the literature. The first comes from the vision statement from a workshop sponsored by NASA and NOAA which states that long-term archiving needs to be a "continuing program for preservation and responsive supply of reliable and comprehensive data, products, and information … for use in building new knowledge to guide public policy and business decisions" [11]. The second definition was presented by John J. Jensen of NOAA/NESDIS at the 2003 IEEE/NASA Mass Storage Conference, as "maintaining the science integrity and long term utility of scientific records" [45].

Both definitions associate scientific stewardship with data preservation as well as access or use in the future. These dual needs are also recognized in the library and records communities (see for example [44] and [46]). Beyond simple access to the original science data, good science stewardship has been shown to allow future development of new or improved products and for use of data in ways that were not originally anticipated [11]. To support these uses however, extensive documentation is needed including complete documentation about the characteristics of the instrument/sensor, its calibration

and how that was validated, the algorithms and any ancillary data used to produce the product, etc. [11] and [12]. This level of associated documentation goes well beyond the typical metadata needs of library or records materials.

4 Data and Metadata Related Challenges

4.1 Open and Proprietary Data and Metadata Formats

The challenges of preserving information for the long term when it is stored in a proprietary format (e.g., MS Word) have been described elsewhere [6]. Commercial pressures do not allow companies to maintain backwards compatibility with each new release for very long. This leaves a very narrow window of opportunity for the information to be migrated to a newer version of the format or a different format, with the attendant risk of loss of functionality or information with each migration.

In the science stewardship realm this may not seem like a large concern since data are still often stored as ASCII tables, flat binary files or one of an increasing number of community standard formats (e.g., shapefiles, HDF-EOS 4). However, much of the associated information about the data - the information that will be needed decades later to allow reanalysis or reprocessing or to allow the development of new products - may very well be stored in a wide variety of proprietary formats (e.g., CAD files, MS Word document).

Even when the data are stored in a non-proprietary format (e.g., CDF, net-CDF or HDF-EOS), the data cannot be maintained forever in their original format. Even so-called standard formats evolve with changes in technology. For example, much of the data stored in the typically petabyte-scale archives of the NASA DAACs, are stored in either HDF-EOS 2.x or HDF-EOS 5.x formats (there are no 3.x or 4.x versions). HDF-EOS 5.x was developed as technological changes mandated entirely new data system architectures incompatible with HDF-EOS 2.x. While tools are available to help users migrate data from the 2.x version to the 5.x version, the new version is not entirely backwards compatible. NASA is currently committed to funding maintenance of both versions [8], but it is not clear whether maintenance will continue once the data are transferred to another agency for long-term archival.

Format evolution can cause particular problems in the Earth sciences where it is necessary to study long data time series in order to detect subtle changes. For example, NSIDC holds brightness temperature and derived sea ice data from a series of passive microwave remote sensing sensors. This is one of the longest continuous satellite remote sensing time series available, dating back prior to 1978. NASA is continuing this time series with a new higher resolution sensor, the Advanced Scanning Microwave Radiometer (AMSR), aboard the Aqua spacecraft. This is an exciting new addition, but scientists and data managers must work to tie the AMSR data into the existing time series. Not only will there be the normal, expected issues of intercallibrating different but related sensors, but someone will likely need to do some data format conversion. The

currently intercallibrated historical data is available in flat binary arrays with ASCII headers while the AMSR data is available in HDF-EOS.

Issues such as these have resulted in a call by some for the establishment of a digital format archive [9], while others have called for conversion to a "Universal Data Format" or other technology-independent-representation upon archival (see for example [10], [13], and [32]). Both of these options require additional research according to a recent NSF-DELOS report [14]. They also increase the need for good metadata describing data format transformations and how these transformations may affect the utility of the data.

4.2 Which Standards and What Metadata?

One of the lessons learned from the ESE experience is that "community-based standards, or profiles of standards, are more closely followed than standards imposed by outside forces" [12]. Developers of the ECS system recognized that having all of the data from the entire suite of satellites and sensors in the same format would simplify user access. After consideration of several potential formats, NASA settled on the HDF-EOS, a derivative of the HDF format standard [8]. A variety of user and producer communities rebelled. As a result, while much of the data stored in the ECS system is stored in HDF-EOS format, there are a number of products, notably the GLAS data stored at NSIDC, that are not in HDF-EOS format.

In addition to the recognition that user community involvement is necessary for successful standards development and adoption, the other important concept from the quote above is the notion of a standards profile, "a specific convention of use of a standard for a specific user community" [12]. It is typically not enough to say that a particular format standard is being used (e.g., HDF or netCDF); it may be necessary to define specific usage conventions possibly even content standards acceptable to a given user community in order to ensure interoperability. These specific conventions or profiles may vary from community to community.

Probably one of the most overworked expressions in the IT industry is "Which standards? There are so many to choose from." It is ironic that not only are there so many standards of a type to choose from; but also that there are so many types of standards about which one must make choices. In the data stewardship realm it is not enough to think about data preservation and data access format standards; one must also think about standards for metadata format and content, documentation format and content, interoperability, etc. For metadata, the question is compounded further by the need to distinguish the type of metadata under discussion, e.g., metadata for data discovery, data preservation, data access, etc.

The Open Archival Information System (OAIS) Reference Model [5] provides an information model (see Figure 1) that describes the different kinds of information needed in order to ingest, preserve and provide access to digital or analog objects. The model appears to be gaining some acceptance in the library and archive communities. It is based

on the concept of an Information Package that can be found by examining its associated Descriptive Information. The components of the Information package itself are:

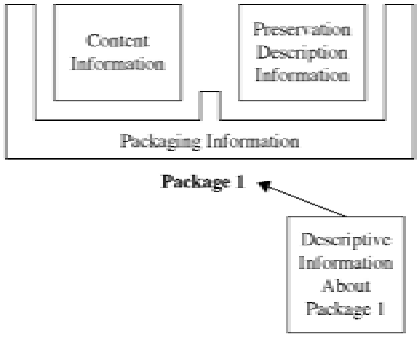


Figure 1: Information Package Components and Relationships [19]

- Content Information-containing both the data object to be preserved as well as enough Representation Information for a targeted user community to understand the data object's structure and content. For science data this would include identification of the data format and any associated profile.

  Of the two components, structure and content, content is more difficult to obtain. The science community has become so specialized that community-specific jargon and underlying assumptions are pervasive. Capturing and documenting these, so that others outside that very small peer group can understand and use the data, is challenging.

  In a very distributed environment, such as the virtual observatories of the future, there will be many thousands or millions of Data Objects preserved in many different places, all of which have the same data format or even the same standard profile. It would be impractical to store the same format information with each object. This may bolster the argument for establishing centralized data format repositories [9], but would require considerable coordination to be successful.

- Preservation Description Information (PDI)-containing the information needed to preserve the object for the long-term. The PDI is comprised of 4 components:

  - Provenance, or the history of the Data Object. In the science arena, this involves tracking the processing history, what input products were used, what version of which algorithms were used, what ancillary information was used for calibration and validation, as well as a host of other instrument-related information. It also includes information about when and where the data were created, processed, and acquired; as well as who was responsible for their creation and what changes have taken place since.

- Reference Information-including information needed to identify this object from the universe of other objects. Much has been written about the need for persistent and unambiguous identifiers (see for example [32], [34], and [15]) and various communities have proposed standards for these (see for example [33] and [13]). A key finding is that in a distributed environment a name issuing authority is needed to prevent naming collisions [32]. In the science community, a hierarchy of identifiers is typically needed. For example, in the ECS system, Earth Science Data Types (ESDT's) are used to identify a data set, or class of objects, while granule ids are used to identify specific objects within the set.

- Fixity Information-documenting the methods used to ensure that the object hasn't been changed in an undocumented manner. This typically includes Cyclic Redundancy Check (CRC) values, digital signature keys, etc. This topic is addressed separately later in this paper.

- Context Information-documenting why this object was created and how it relates to other objects.

While the OAIS reference model discusses the types of metadata or information that must be gathered in order to preserve an object, it leaves the actual definition of that metadata to the individual archive or archive group. Several groups within the library community have independently developed their own preservation metadata specifications (see [21], [22], [23], and [24]) and have recently come together under the joint auspices of the Research Libraries Group (RLG) and the Online Computer Library Center (OCLC) to develop an OAIS-based metadata framework that "could be readily applied to a broad range of digital preservation activities" [25].

While providing a useful starting point for the science community, the OCLC/RLG framework is not adequate for preserving science data. The science community is typically more interested in preserving information about how the data were created than in preserving any particular presentation mechanism. This is to be expected given the different kinds of uses to which patrons of libraries and science users put the materials accessed. Typically a library patron expects to experience the material using his or her senses; to read, listen to, touch, or watch; but not to transform the materials accessed. Scientists typically access data so that it can be transformed, analyzed, used as an input to a model or new product, compared with other data, etc. Changes in presentation format over time as technology, programming languages, and visualization tools change, are not that important – the important things are the original bits and their meaning.

In the earth science realm probably the most relevant metadata standard is the "Content Standard for Digital Geospatial Metadata" established by the Federal Geographic Data Committee (FGDC) [19]. President Clinton mandated that federally funded geospatial data (i.e., most Earth science data including ESE data) adhere to the FGDC standard in an 11 April 1994 executive order [20]. The FGDC standard "was developed from the perspective of defining the information required by a prospective user to determine the

availability of a set of geospatial data; to determine the fitness and the set of geospatial data for an intended use; to determine the means of accessing the set of geospatial data; and to successfully transfer the set of geospatial data" [19]. As such, there is some but not complete overlap with the kinds of metadata called for by the OAIS reference model. Much of the preservation metadata called for by the OAIS model is not part of the FGDC standard.

In the international standards realm, the equivalent to the FGDC standard is the ISO 19115 standard [26]. Like the FGDC standard, the ISO standard is meant to facilitate, discovery, assessment for use, access and use and, like the FGDC standard, does not address much of the preservation metadata of the OAIS reference model. The FGDC has developed a draft "cross-walk" between the FGDC and ISO 19115 standards which will help FGDC-compliant users also become ISO 19115 compliant users.

Both the FGDC and ISO 19115 standards are content standards, not implementation standards, yet organizations must choose implementation options. Consensus seems to be building that Extensible Markup Language (XML) should be the implementation standard for metadata. ISO Technical Committee 211 is in the process of developing an UML implementation standard for the ISO 19115 metadata standard that will include an associated XML schema. XML is also the recommendation of the National Research Council [2].

4.3 Preservation vs. Access

Users want data in formats that are easy to use. The desired format may be a community-based standard, or it may be a format that is compatible with other related data sets. Furthermore, our experience at NSIDC shows that users usually need spatial or temporal subsets of large collections and may need to combine several products. They may also need the data in a different grid or projection than the original data. In other words, the utility or essence of science data is not strongly associated with any particular access format. Indeed, many different formats, grids, and projections may need to be supported at any given time. This is significantly different from other disciplines concerned with digital preservation where it is often essential to preserve the essence of the original experience for multimedia digital objects such as movies, applications, or interactive web sites. In the Earth science community it makes more sense to consider preservation and access formats independently. Access formats are likely to change quickly over time, while preservation formats should be more stable.

There are similar issues with preservation and access metadata. There are advantages to preserving the metadata with the actual data (see section 4.4), but much of the metadata is only relevant to the archivist. The preservation-specific metadata should probably be separated from the data upon delivery to the user to minimize user confusion.

Some have called for completely separate storage of preservation and access data [13]. However, storage of multiple copies of the data is unaffordable with large data sets or when there are multiple access formats. In many cases, the issue may be how to afford preservation of even a single copy of the data! There is some agreement that the best

CD contained over 50 data sets and nearly 100 references to other data sets held by different "nodes" of the GGD system. Unfortunately, funding for the GGD did not continue past 1998. It wasn't until 2002, when a new initiative started to create an updated version of the CD (now a three CD set [42]), that any maintenance of the data from the 1998 version resumed. Regrettably, dozens of the distributed or "brokered" products were no longer readily available. NSIDC has plans to try and track down or "rescue" some of these data sets, but the four- to five-year time lag and the globally distributed nature of the data sets will make it very challenging. This illustrates the need for nearly constant tracking of distributed data to ensure its continued availability, or a clear and usable means (with incentives) for providers to provide updates to any central access point.

4.5 Data Security and Integrity

Ultimately, keeping track of data and metadata becomes an issue of data integrity. Scientists need to trust the validity of the data they use. They need to know that the data came from a scientifically reputable source and that the data have not been corrupted in any way.

Scientific integrity is an ill-defined concept, but it is rooted in the scientific method. Experiments must be repeatable. Results from experiments should be published in peer-reviewed literature. The data and information used in the experiment must be specifically acknowledged and generally accessible when possible. Traditionally this is handled in the literature through a system of formal citations. But while methods for citing information sources are well established and traceable, methods for citing data sources are more variable. Historically, with small non-digital data sets, the data may have been published directly in a journal or monograph that could specifically be cited. This was not an entirely consistent process, though, and as data sets have grown, authors have adopted different methods for acknowledging their data sources. Some authors may provide a simple acknowledgement of the data source in the body of an article or the acknowledgements section. Other authors may cite an article published by the data provider that describes the data set and its collection.

As publishers of data, we at NSIDC have found these historical approaches lacking. General data acknowledgements are difficult to trace, are often imprecise, and sometimes do not acknowledge the true data source. For example, an acknowledgement of "NSIDC's SSM/I sea ice data" could actually refer to one of several different data sets and it makes no reference to the actual scientists who developed the sea ice algorithm. Citing a paper about the data is better, but in many cases such papers may not exist, they may only describe a portion of the data set, or their description may not be relevant to the new application of the data. In any case, it is not clear how to actually acquire the data—a necessary step if an experiment is to be repeated. We recommend that users cite the actual data set itself, much as they would a book or journal article. The "author" is typically the data provider or the person who invested intellectual effort into the creation of the data set (e.g., by creating an algorithm), while NSIDC or other archive that

CD contained over 50 data sets and nearly 100 references to other data sets held by different "nodes" of the GGD system. Unfortunately, funding for the GGD did not continue past 1998. It wasn't until 2002, when a new initiative started to create an updated version of the CD (now a three CD set [42]), that any maintenance of the data from the 1998 version resumed. Regrettably, dozens of the distributed or "brokered" products were no longer readily available. NSIDC has plans to try and track down or "rescue" some of these data sets, but the four- to five-year time lag and the globally distributed nature of the data sets will make it very challenging. This illustrates the need for nearly constant tracking of distributed data to ensure its continued availability, or a clear and usable means (with incentives) for providers to provide updates to any central access point.

4.5 Data Security and Integrity

Ultimately, keeping track of data and metadata becomes an issue of data integrity. Scientists need to trust the validity of the data they use. They need to know that the data came from a scientifically reputable source and that the data have not been corrupted in any way.

Scientific integrity is an ill-defined concept, but it is rooted in the scientific method. Experiments must be repeatable. Results from experiments should be published in peer-reviewed literature. The data and information used in the experiment must be specifically acknowledged and generally accessible when possible. Traditionally this is handled in the literature through a system of formal citations. But while methods for citing information sources are well established and traceable, methods for citing data sources are more variable. Historically, with small non-digital data sets, the data may have been published directly in a journal or monograph that could specifically be cited. This was not an entirely consistent process, though, and as data sets have grown, authors have adopted different methods for acknowledging their data sources. Some authors may provide a simple acknowledgement of the data source in the body of an article or the acknowledgements section. Other authors may cite an article published by the data provider that describes the data set and its collection.

As publishers of data, we at NSIDC have found these historical approaches lacking. General data acknowledgements are difficult to trace, are often imprecise, and sometimes do not acknowledge the true data source. For example, an acknowledgement of "NSIDC's SSM/I sea ice data" could actually refer to one of several different data sets and it makes no reference to the actual scientists who developed the sea ice algorithm. Citing a paper about the data is better, but in many cases such papers may not exist, they may only describe a portion of the data set, or their description may not be relevant to the new application of the data. In any case, it is not clear how to actually acquire the data—a necessary step if an experiment is to be repeated. We recommend that users cite the actual data set itself, much as they would a book or journal article. The "author" is typically the data provider or the person who invested intellectual effort into the creation of the data set (e.g., by creating an algorithm), while NSIDC or other archive that

distributed the data might be considered the publisher. It is also crucial to include publication dates to distinguish between different versions of related data sets. In any case, we try and provide a specific recommended citation for every data set we distribute. Although we have met some sporadic resistance from occasional providers who wish only for their papers to be cited, this approach has become broadly accepted. It is the approach specifically recommended by the International Permafrost Association ([41], [42]), and has generally been accepted by the other NASA DAACs.

This formal citation approach works well when there is a clear and reputable data publisher even in a distributed environment. But the distributed environment may provide additional challenges, especially if data sources are somewhat ephemeral or hard to identify. For example, in a peer-to-peer system, the access mechanism needs to specifically identify the different peers and possibly some assessment of their stability. This is somewhat different than peer-to-peer systems in other areas such as music where users generally don't care where the music came from as long as it is the piece they wanted. With the rise of electronic journals we have also heard informal discussion of including the actual data used in the publication itself. Although this approach obviously includes many of the same data preservation challenges already discussed, it is an intriguing concept worthy of further exploration.

Once the scientific integrity of a data set has been assured, assurance is needed that the data received is what was expected. Several authors discuss the use of public/private key cryptography and digital signatures as methods for ensuring the authenticity of the data (see for example [35] and [36]). Lynch points out that we know very little about how these technologies behave over very long times and that, as a consequence, information about evolution of these technologies will likely be important to preserve [37].

For a scientist to be able to trust that the data have not been changed the scientist must be able to trust that the preservation practices of the source of the data are adequate: that archive media are routinely verified and refreshed, that the facilities are secure, that processes to verify and ensure the fixity of the data are operational, that geographically distributed copies of the data are maintained as a protection against catastrophe, and that disaster recovery plans and procedures are in place. To verify these practices, the RLG/OCLC Working Group on Digital Archive Attributes suggests that a process for certification of digital repositories be put in place [34]; while Ashley suggests that administrative access to data and metadata be "subject to strong proofs of identity" [36]. Once again, a distributed data environment may make implementing these suggestions more difficult.

4.6 Long-Term Preservation and Technology Refresh

A continual theme in this paper is how the speed of technological change presents a major challenge for preserving data over the long term. As a recent report by the National Science Foundation and the Library of Congress puts it "digital objects require constant and perpetual maintenance, and they depend on elaborate systems of hardware,

data user community. A good data manager, equipped with the right tools, should be working closely with the data provider to uncover any known limitations in the data.

For example, it may be self evident to developers of sea ice detection algorithms for passive microwave remote sensing that their methodology is well-suited to detection of trends in ice concentration over time but ill-suited for representing precise local ice conditions on a given day. This may not be apparent to a biologist who uses a near-real time passive microwave derived product to associate sea ice conditions in the Beaufort Sea with polar bear migration. While this is an extreme example, it highlights the need for scientists and data managers to work closely together to carefully document and track new and potentially unexpected uses of the data. It is also important to realize that the risks of inappropriate data applications could increase over time.

Of course data can also be improved. New algorithms, new calibration methods, and new instruments may be developed. In Earth science in particular, it is important to detect variability over long time periods. This means that different instruments must be intercallibrated to ensure a consistent time series, i.e. we need to be able to ensure that any changes we detect in a data stream result from actual geophysical processes not just changes in instruments or algorithms. This again requires collaboration between the data manager and the scientist. This is certainly possible in distributed environments, but mechanisms should be established to ensure that information about data harmonization and improvements are readily available to users. Traditionally, this was the role of the data steward or data manager (see, for example, [29]). It is less clear how this would work in a distributed environment, but knowledge bases and data mining systems are likely to contribute.

On a related note, to ensure maximum scientific understanding of an issue, data and support services need to be readily available to as many users as possible [11]. This is necessary to ensure all possible scientific ideas are explored and that scientific experiments can be duplicated. The necessary broad access may be better realized in a distributed data model, but only if the challenges in section four are addressed. Again this will require close interaction with the users.

Historically, NSIDC has addressed these scientific issues by working closely with its data providers and by having scientific data users and developers on staff. This becomes a less practical approach in a distributed data environment where data may be held and distributed by individuals and institutions with varying levels of scientific and data management expertise. It will become increasingly important to formally address the relationship of data managers and scientists as new distributed data management models are developed.

5.2 Decisions, Decisions, Decisions - Deciding What Data to Acquire and Retain

One of the most difficult decisions in data archival is which data to acquire and keep and which data to throw away. Although, there is still no effective business model that

data user community. A good data manager, equipped with the right tools, should be working closely with the data provider to uncover any known limitations in the data.

For example, it may be self evident to developers of sea ice detection algorithms for passive microwave remote sensing that their methodology is well-suited to detection of trends in ice concentration over time but ill-suited for representing precise local ice conditions on a given day. This may not be apparent to a biologist who uses a near-real time passive microwave derived product to associate sea ice conditions in the Beaufort Sea with polar bear migration. While this is an extreme example, it highlights the need for scientists and data managers to work closely together to carefully document and track new and potentially unexpected uses of the data. It is also important to realize that the risks of inappropriate data applications could increase over time.

Of course data can also be improved. New algorithms, new calibration methods, and new instruments may be developed. In Earth science in particular, it is important to detect variability over long time periods. This means that different instruments must be intercallibrated to ensure a consistent time series, i.e. we need to be able to ensure that any changes we detect in a data stream result from actual geophysical processes not just changes in instruments or algorithms. This again requires collaboration between the data manager and the scientist. This is certainly possible in distributed environments, but mechanisms should be established to ensure that information about data harmonization and improvements are readily available to users. Traditionally, this was the role of the data steward or data manager (see, for example, [29]). It is less clear how this would work in a distributed environment, but knowledge bases and data mining systems are likely to contribute.

On a related note, to ensure maximum scientific understanding of an issue, data and support services need to be readily available to as many users as possible [11]. This is necessary to ensure all possible scientific ideas are explored and that scientific experiments can be duplicated. The necessary broad access may be better realized in a distributed data model, but only if the challenges in section four are addressed. Again this will require close interaction with the users.

Historically, NSIDC has addressed these scientific issues by working closely with its data providers and by having scientific data users and developers on staff. This becomes a less practical approach in a distributed data environment where data may be held and distributed by individuals and institutions with varying levels of scientific and data management expertise. It will become increasingly important to formally address the relationship of data managers and scientists as new distributed data management models are developed.

5.2 Decisions, Decisions, Decisions - Deciding What Data to Acquire and Retain

One of the most difficult decisions in data archival is which data to acquire and keep and which data to throw away. Although, there is still no effective business model that

demonstrates the costs and benefits of long-term data archival [15], it is clearly impractical to keep all data for all time. That said, we need to recognize that many data sets often have unexpected future applications (see [11] for examples). A simple approach would be to archive a very low level of the data along with the necessary algorithms to process the higher level products. However, this must be viewed only as a minimum since it does not allow for the necessary simple and broad access described above.

It is probably not possible to describe any one infallible data acquisition and deposition scheme. However, any data stewardship model must explicitly include a method for development of such a scheme for different types of data and user communities. These schemes must explicitly include knowledgeable and experienced users of the data who are directly involved in generating new products and data quality control [11].

5.3 Upfront Planning

Our experience at NSIDC has shown that by working with the scientists and data providers early in an experiment or mission, ideally before any data are actually collected, we can significantly improve the quality and availability of the data. Most scientists can probably think of a field campaign where the data are no longer available. NSIDC worked to avoid this problem by working closely with the investigators conducting the Cold Land Processes field experiment in the Colorado Rocky Mountains during the winter and spring of 2002 and 2003 (see [30]). Not only was NSIDC involved in the planning of the data collection, but also provided data technicians who worked closely with field surveyors during the experiment. These data technicians learned the data collection protocol with the surveyors, helped collect some of the data, and entered the data into computers the night after they were collected. By learning the protocol and immediately entering the data, technicians were able to identify missing values and anomalies in the data and run some automated quality control checks. They were then able to follow up with the surveyors soon after they collected the data to correct specific problems and to improve later data collection. Technicians were also able to provide the data to the lead scientists for immediate assessment. Overall, this led to a 10 to 20 percent improvement in data quality [31].

NSIDC has had similar experience with recent satellite remote-sensing missions. NSIDC is the archive for all the data from NASA's Advanced Microwave Scanning Radiometer (AMSR) and Global Laser Altimetry System (GLAS). Although NSIDC was not directly involved in the acquisition of the data, it did work closely with the mission science and instrument teams well before the instruments were even launched. This allowed the data managers to have a much greater understanding of the engineering aspects of the data and the algorithms used to produce the higher-level products. The result is much better documentation and much earlier data availability. Data from both of these missions were available to the public only months after launch, in contrast to years with some historical systems where data managers were not involved until well after their launch (e.g, sea ice data from SSM/I).

There is nothing inherent about distributed data systems that should preclude early involvement of data managers, but again this is something to consider in the design of those systems. Furthermore, data manager involvement could be more difficult if traditional data management organizations are not directly involved in the distributed data system.

6 Conclusions

The scientific method requires that experimental results be reproducible. That means the data used in the original experiment must be available and understandable. Furthermore, reexamination of an early data set often can yield important new results.

Maintaining access to and understanding of scientific data sets has been a challenge throughout history. The trend to a more geographically distributed data management model may improve data access in the short run but raises additional challenges. We should be able to address many of these challenges by developing new tools and data management systems, but we must not forget the human component. Experience and a review of the known data management issues show that we achieve the greatest success in long term data stewardship only when there is a close collaboration between data providers, data users, and professional data stewards. As we move forward, we need to ensure that new technologies and new data archive models enhance this collaboration.

[1] NOAA's National Geophysical Data Center. "About the World Data Center System." December 29, 2003. http://www.ngdc.noaa.gov/wdc/about.shtml. January 2004.

[2] National Research Council. 2003. "Government Data Centers: Meeting Increasing Demands." The National Academies Press.

[3] Ad Age Group. "Data Center." January 6, 2004. http://www.adage.com/datacenter.cms. Data Center. January 6, 2004.

[4] Texas Cancer Council. "Texas Cancer Data Center." December 23, 2003. http://www.txcancer.org/. Texas Cancer Data Center. January 6, 2004.

[5] CCSDS. 2002. "Reference Model for an Open Archival Information System (OAIS)." CCDSD 650.0-B-1. Blue Book. Issue 1. January 2002. [Equivalent to ISO 14721:2002].

[6] Barnum, George D. and Steven Kerchoff. "The Federal Depository Library Program Electronic Collection: Preserving a Tradition of Access to United States Government Information." December 2000. http://www.rlg.org/events/pres-2000/barnum.html. January 7, 2003.

[7] Moore, Reagan W. October 7, 1999. "Persistent Archives for Data Collections." SDSC Technical Report sdsc-tr-1999-2.

[8] Ullman, Richard. "HDF-EOS Tools and Information Center." 2003. http://hdfeos.gsfc.nasa.gov/hdfeos/index.cfm. January 7, 2004.

[9] Abrams, Stephen L. and David Seaman. "Towards a global digital format registry." World Library and Information Congress: 69th IFLA General Conference and Council, Berlin, August 1-9, 2003 http://www.ifla.org/IV/ifla69/papers/128e-Abrams_Seaman.pdf

[10] Shepard, T. and D. MacCarn. 1999. *The Universal Preservation Format: A Recommended Practice for Archiving Media and Electronic Records*. WGBH Educational Foundation. Boston. 1999.

[11] Hunolt, Greg. *Global Change Science Requirements for Long-Term Archiving. Report of the Workshop, Oct 28-30, 1998.* USGCRP Program Office. March 1999.

[12] SEEDS Formulation Team. Strategic Evolution of Earth Science Enterprise Data Systems (SEEDS) Formulation Team final recommendations report. 2003. http://lennier.gsfc.nasa.gov/seeds/FinRec.htm. Jan. 2004.

[13] The Cedars Project. "Cedars Guide to The Distributed Digital Archiving Prototype." March 2002. http://www.leeds.ac.uk/cedars/guideto/cdap/. December 2003.

[14] "Invest to Save: Report and Recommendations of the NSF-DELOS Working Group on Digital Archiving and Preservation." 2003. Prepared for the National Science Foundation's (NSF) Digital Library Initiative and the European Union under the Fifth Framework Programme by the Network of Excellence for Digital Libraries (DELOS).

[15] "It's About Time: Research Challenges in Digital Archiving and Long-Term Preservation, Final Report, Workshop on Research Challenges in Digital Archiving and Long-Term Preservation." August 2003. Sponsored by the National Science Foundation, Digital Government Program and Digital Libraries Program, Directorate for Computing and Information Sciences and Engineering, and the Library of Congress, National Digital Information Infrastructure and Preservation Program, August 2003.

[16] Lachman, B.E., A Wong, D. Knopman, and K. Gavin. 2002. "Lessons for the Global Spatial Data Infrastructure: International case study analysis." Santa Monica, CA: RAND.

[17] Rhind, D. 2000. Funding an NGDI. In Groot, R. and J. McLaughlin, eds. 2000. *Geospatial data and infrastructure: Concepts, cases, and good practice*. Oxford University Press.

[18] PDG (Panel on Distributed Geolibraries, Mapping Science Committee, National Research Council). 1999. *Distributed geolibraries: Spatial information resources*. Washington, DC: National Academy Press.

[19] Federal Geographic Data Committee. Revised June 1998. "Content Standard for Digital Geospatial Metadata." Washington, D.C.

[20] Clinton, W. 1994. *Coordinating geographic data acquisition and access to the National Spatial Data Infrastructure, Executive Order 12906*. Washington, DC. Federal Register 59 17671-4. 2pp.

[21] The CEDARS Project. 2001. "Reference Model for an Open Archival Information System (OAIS)." http://www.ccds.org/documents/pdf/CCSDS-650.0-R-2.pdf. December 2003.

[22] National Libarary of Australia. 1999. "Preservation Metadata for Digital Collections." http://www.nla.gov.au/preserve/pmeta.html". December 2003.

[23] NEDLIB. 2000. "Metadata for Long Term Preservation." http://www.kb.nl/coop/nedlib/results/preservationmetadata.pdf. December 2003.

[24] OCLC. 2001. "Preservation Metadata Element Set – Definitions and Examples." http://www.oclc.org/digitalpreservation/archiving/metadataset.pdf. December 2003.

[25] The OCLC/RLG Working Group on Preservation Metadata. 2002. "Preservation and the OAIS Information Model – A Metadata Framework to Support the Preservation of Digital Objects." OCLC Online Computer Library Center, Inc.

[26] ISO Technical Committee ISO/TC 211, Geographic Information/Geomatics. "May, 2003. "Geographic information – Metadata. ISO 19115:2003(E)." International Standards Organization.

[27] ISO Technical Committee ISO/TC 200. November 2002. "Scope." http://www.isotc211.org/scope.htm#19139, January 2003.

[28] Holdsworth, D. "The Medium is NOT the Message or Indefinitely Long-Term Storage at Leeds University." 1996. http://esdis-it.gsfc.nasa.gov/MSST/conf1996/A6_07Holdsworth.html. January 2004.

[29] Stroeve, J, X. Li, and J. Maslanik. 1997. "An Intercomparison of DMSP F11- and F13-derived Sea Ice Products." NSIDC special report 5. http://nsidc.org/pubs/special/5/index.html. January 2004.

[30] Cline, D. et al. 2003. Overview of the NASA cold land processes field experiment (CLPX-2002). *Microwave Remote Sensing of the Atmosphere and Environment III*. Proceedings of SPIE. Vol. 4894.

[31] Parsons, M. A., M. J. Brodzik, T. Haran, N. Rutter. 2003. *Data management for the Cold Land Processes Experiment*. Oral presentation, 11 December 2003 at the meeting of the American Geophysical Union.

[32] The CEDARS Project. "CEDARS Guide to: Digital Preservation Strategies." April 2, 2002. http://www.leeds.ac.uk/cedars/guideto/dpstrategies/dpstrategies.html. January 2004.

[33] National Library of Australia. "Persistent identifiers – Persistent identifier Scheme Adopted by the National Library of Australia." September 2001. http://www.nla.gov.au/initiatives/nlapi.html. January 2004.

[34] RLG/OCLC Working Group on Digital Archive Attributes. May 2002. "Trusted Digital Repositories: Attributes and Responsibilities." RLG Inc.

[35] Brodie, N. December 2000. "Authenticity, Preservation and Access in Digital Collections." Preservation 2000: An International Conference on the Preservation and Long Term Accessibility of Digital Materials. RLG Inc.

[36] Ashley, K. December 2000. "I'm me and you're you but is that that?" Preservation 2000: An International Conference on the Preservation and Long Term Accessibility of Digital Materials. RLG Inc.

[37] Lynch, C. 2000. "Authenticity and Integrity in the Digital Environment: An Exploratory Analysis of the Central Role of Trust." Council on Library and Informational Resources, Washington D.C.

[38] Thibodeau, K. 2002. "Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years." The State of Digital Preservation: An International Perspective. Conference Papers and Documentary Abstracts. http://www.clir.org/pubs/reports/pub107/thibodeau.html. December 2003.

[39] Lee, K., Slattery, O., Lu, R., Tang X., McCrary V. 2002. *The State of the Art and Practice in Digital Preservation.* J. Res. Natl. Inst. Stand. Technol. 107, 93-106.

[40] DoD 8320.1-M. "Data Administration Procedures." March 29, 1994. Authorized by DoD Directive 8320.1. September 26, 1991. *Reference*: DoD 8320.1-M-1, "Data Element Standardization Procedures," January 15, 1993.

[41] International Permafrost Association, Data and Infromation Working Group, compilors. 1998. *Circumpolar active-layer permafrost system, version 1.0.* Boulder, CO: National Snow and Ice Data Center/World Data Center for Glaciology. CD-ROM.

[42] International Permafrost Association Standing Committee on Data Information and Communication, compilors. 2003. *Circumpolar active-layer permafrost system,*

version 2.0. Edited by M. Parsons and T. Zhang. Boulder, CO: National Snow and Ice Data Center/World Data Center for Glaciology. CD-ROM.

[43] Diomidis Spinellis. The decay and failures of web references. 2003. Communications of the ACM, 46(1):71-77.

[44] ICTU. "Digital Preservation Testbed." Digitale Duurzaamheid. 2004. http://www.digitaleduurzaamheid.nl/home.cfm Jan. 2004.

[45] Diamond, H., Bates, J., Clark D., Mairs R. "Archive Management – The Missing Component." April 2003, http://storageconference.org/2003/presentations/B06_Jensen..pdf, NOAA/NESDIS. January 2004.

[46] Hedstrom, M. 2001. "Digital Preservation: A Time Bomb for Libraries." http://www.uky.edu/~kiernan/DL/hedstrom.html. Jan. 2004.

NARA's ELECTRONIC RECORDS ARCHIVES (ERA) – THE ELECTRONIC RECORDS CHALLENGE

Mark Huber
American Systems Corp.
National Archives and Records Administration
8601 Adelphi Rd., Rm. 1540, College Park, MD 20740
Tel: +1-301-837-0420
mark.huber@nara.gov

Alla Lake
Lake Information Systems, LLC
National Archives and Records Administration
8601 Adelphi Rd., Rm. B550, College Park, MD 20740
Tel: +1-301-837-0399
alla.lake@nara.gov

Robert Chadduck
National Archives and Records Administration
8601 Adelphi Rd., Rm. 1540, College Park, MD 20740
Tel: +1-301-837-0394
Robert.chadduck@nara.gov

Abstract
The National Archives and Records Administration (NARA) is the nation's recordkeeper. NARA is a public trust that safeguards the records of the American people, ensuring the accountability and credibility of their national institutions, while documenting their national experience. Today NARA holds an estimated 4 billion records nationwide. The Archives consists of the permanently valuable records generated in all three branches of the Federal Government. These record collections span this country's entire experience, across our history, the breadth of our nation, and our people. While paper documents presently predominate, NARA holds enormous numbers of other media, such as reels of motion picture film, maps, charts, and architectural drawings, sound and video recordings, aerial photographs, still pictures and posters, and computer data sets. It is that last medium, the electronic records, that is the fastest growing record keeping medium in the United States and elsewhere in the world. Since 1998, NARA has established key partnerships with Federal Agencies, state and local governments, universities, other national archives, the scientific community, and private industry to perform research enabling better understanding of the problems and the possibilities associated with the electronic records challenge. The challenge of electronic records encompasses the proof and assurance of records authenticity and assurance of record persistence and ready access to records over time.

1. Background/General Project Description

"Electronic records pose the biggest challenge ever to record keeping in the Federal Government and elsewhere. There is no option to finding answers…the alternative is irretrievable information, unverifiable documentation, diminished government accountability, and lost history."

John Carlin, The Archivist of the United States

The National Archives and Records Administration (NARA) is the nation's recordkeeper. NARA is a public trust that safeguards the records of the American people, ensuring the accountability and credibility of their national institutions, while documenting their national experience. Pursuant to legislation codified under Title 44 of the United States Code the Archivist of the United States has authority to provide guidance direction and assistance to Federal officials on the management of records, to determine the retention and disposition of records, to store records in centers from which agencies can retrieve them, and to take into the archival facilities of the National Archives and Presidential libraries, for public use, records that he determines "to have sufficient historical or other value to warrant their continued preservation by the United States Government." (44 U.S.C. 2107) Similarly, under the Presidential Records Act, when a President leaves office, the Archivist of the United States assumes responsibility "for the custody, control, and preservation of, and access to, the Presidential records of that President". Both the Government and the public rely on NARA to provide this and subsequent generations of the American public with access to extraordinarily high accretion rate, increasingly diverse, and arbitrarily complex collections of historically valuable federal, presidential and congressional electronic records collections.

The technology challenge confronting NARA is repeatedly confirmed as among the President's research priorities. In the supplement to the President's budget for fiscal year 2004, The National Science and Technology Council expressly acknowledges that "R&D in advanced technologies that enable preservation and utility of electronic information archives…," and "…digital archives of core knowledge for research and learning" is "far from finished." Especially prominent is the Council's explicit identification of "….substantial technical issues – such as interoperability among file formats, indexing protocols, and interfaces; data management, storage and validation; … and long term preservation – that impede development of digital libraries…" Similarly noted is research enabling agencies to move "…toward two ambitious goals: quick, easy, and secure on-line access for citizens to government services and information, and radical reduction in internally duplicative record-keeping, ... through coordinated development of IT standards and procedures..." [1]

Experts predicted in FY2003 that electronic records volumes will swell by orders of magnitude over this decade, presenting enormous challenges for society along with unprecedented opportunities for U.S. advanced research and technological innovation.", …fused with requirements for… "technologies for rapid mining, filtering, correlating and assessing of vast quantities of heterogeneous and unstructured data", and… "tools for collecting, archiving and synthesis." [2]

Similarly, among the president's FY2002 research priorities:

"Strategies to assure long-term preservation of digital records constitute another particularly pressing issue for research. As storage technologies evolve with increasing speed to cope with the growing demand for storage space, the obsolescence of older storage hardware and software threatens to cut us off from the electronically stored past." [3]

The Archivist is authorized by law to "conduct research with respect to the improvement of records management practices and programs." [44 U.S.C Section 2904(c)(2)]. Since 1998, NARA has established key partnerships with Federal Agencies, state and local governments, universities, other national archives, the scientific community, and private industry to perform research enabling better understanding of the problems and the possibilities associated with the electronic records challenge.

NARA's Key Research Partners

- National Science Foundation (NSF)
- San Diego Supercomputer Center (SDSC)
- University of Maryland (UMd)
- Georgia Tech Research Institute (GTRI)
- U.S. Army Research Laboratory (ARL)
- National Computational Science Alliance (NCSA)
- National Institute of Standards of Technology (NIST)
- National Nuclear Security Administration (NNSA)
- National Aeronautics and Space Administration (NASA)
- U.S. Department of Defense (DOD)
- Library of Congress (LC)
- International Research on Permanent Authentic Records in Electronic Systems (InterPARES)
- Digital Library Federation (DLF)
- Global Grid Forum (GGF)

NARA's ERA Program includes ongoing sponsorship, support, and collaboration in technology research activities relevant to developing and sustaining the systematic capability for transfer, preservation, and sustained access to electronic records. ERA must be dynamic in response to continuing technology evolution, ensuring that electronic records delivered to future generations of Americans are as authentic decades in the future as they were when first created.

Among the findings presented in the report of the Committee on Digital Archiving and the National Archives and Records Administration of the Computer Science and Telecommunications Board (CSTB) for the National Research Council of the National Academies are findings that while no turnkey system, application, or product exists in the marketplace which meets NARA's requirements, the system can and should be built. [4]

2. Program Status

In response to the digital records challenge, Congress, in November 2001, acting through the Treasury and General Government Appropriations Act {P.L.107-67}, approved the fiscal 2002 budget that included $22.3 million for Electronic Records Archives (ERA) Program. Similarly, in January 2003, Congress, acting through the Consolidated Appropriation Resolution, 2003 {P.L.108-7}, approved the fiscal 2003 budget that included $11.8 million for the Electronic Records Archives (ERA) Program. At the time of this writing, and while the final appropriations have not passed, both the House of Representatives and the Senate have agreed to fund the ERA Program at the $35.7M level in the President's FY2004 request. The official Request for Proposal (RFP) for the ERA system was released to the public on December 5, 2003. At the time of the RFP release, proposals from industry were required to be submitted to NARA by January 28, 2004. The ERA program schedule calls for up to two contract awards to be made by mid-2004.

NARA has structured the ERA procurement to fundamentally be a challenge to industry to propose innovative ways to address the challenges represented by the large number and variety of electronic records generated and used by the Federal government. The ERA procurement strives to define the electronic records challenge without prescribing implementations or techniques with which to address the issues. Again, NARA wants to engage industry in crafting long term responses to the various technical and operational issues that ERA represents. This paper goes on to explore some of the archival, technical, and operational issues that the ERA program sees as important to the success of ERA.

3. Goals, Issues, and Challenges for Electronic Records - Persistence, obsolescence, access over time

Today NARA holds in the National Archives of the United States and the Presidential Libraries an estimated 4 billion records nationwide. The archives consist of the permanently valuable records generated in all three branches of the Federal Government, supplemented with donated documentary materials. [5]

These records span this country's entire experience, across our history, the breadth of our nation, and our people. Not surprisingly, with the passage of time, the medium of the records of the United States has become diverse in format. While paper documents presently predominate, NARA holds enormous numbers of

- reels of motion picture film,
- maps, charts, and architectural drawings,
- sound and video recordings,
- aerial photographs,
- still pictures and posters, and
- computer data sets. [6]

It is that last medium – computer data sets - the electronic records, that is the fastest growing record keeping medium in the United States and elsewhere in the world. According to the *How Much Information? 2003* study from the University of California

Berkley School of Information Management and Systems, released in October 2003, the worldwide production of original information stored digitally on magnetic media has grown by 80% in the time elapsed between the 1999 and 2002 samples. The upper boundary study volume estimate in that category of information for 1999 was 2.8 Peta Bytes and for 2002 – 4.99 Peta Bytes. [7]

The digital (electronic) storage of information –has been growing in proportion to the rise in creation and use of information in general. There is no consensus optimal method for the long term preservation of electronic records. A number of approaches are being used in the industry singly and in combination. Each of the approaches brings with it its own cost, as well as operational and reliability concerns. The larger the size of the electronic records holdings, the more important it is to carefully select and design the preservation approach.

Preserving electronic records serves the same fundamental *purpose* as preserving any other type of record: to enable the records to continue to provide evidence and information about the decisions, acts, and facts described in the records with the same degree of reliability as when the record was created. However, the *process* of preserving electronic records is substantially different than the preservation of traditional, non-electronic records. Traditional records are aptly termed "hard copy" in that the information that the record contains is inscribed in a hard, indissoluble manner on a physical medium, and the physical inscription conveys the information the record is intended to provide. Therefore, preservation traditionally focused on the physical object. However, an electronic record is inscribed on a physical medium as a sequence of binary values which must always be translated into a different form – the form of a record – in order to communicate the information the record was meant to convey. Therefore, preserving an electronic record requires maintaining the ability to reproduce that record from stored data. While the preservation of a paper record can be deemed successful if that record remains physically intact in storage, the success of a process of preserving an electronic record can only be verified by translating the stored bits into the form of the record. It is the result of this reproduction, not the stored bits, that literally **is** the electronic record. If the wrong process is applied, or if the process is not executed correctly, the result will not be an authentic copy of the record. Over time, reproducing an electronic record is challenging because the conventions for representing information in digital form change along with hardware and software. Newer systems may not be able to process older formats, or may do so incorrectly. [8]

Archiving of electronic records brings with it an increased challenge of authenticity of the record and a more difficult burden of proof of that authenticity. For the ERA program, electronic record *authenticity* is defined as *the property of a record that it is what it purports to be and has not been corrupted.* Given the legal, historical, and cultural significance of national or institutional record holdings, authenticity of the records is essential. Establishing authenticity of a paper, photographic negative, or other physical medium-based record has historically been accomplished by establishing that the record itself is, or is based on, an original via the proof that the medium of the record (or the medium of the basis record) is the original and there is a clear chain of custody

associated with the record. Stringent requirements assigned to electronic record collections to support a continuing burden of proof relates to the attainment of criteria for authenticity over time. Electronic records present special challenges with respect to the proof of record authenticity as the record is preserved over time due to the both increased risk of corruption of the record when it exists in digital form.

Records are being created in progressively larger volumes through the use of electronic hardware and the associated software applications. Some of the records are traditional textual or graphic documents that could have been originated with the use of pen and paper. At least in theory, their content can be preserved in hard copy. The bulk of them, however, are in most respects indelibly tied to the technology that produced them, such as the contents of data base systems, interactive Web pages, geographic information systems, and virtual reality models. [9] These later types of electronic records need to be preserved in electronic form in order to preserve the essential properties of the record other than pure content - the context, structure, and behavior. Whenever a mix of technologies are involved in the creation, maintenance, and presentation of the record, preservation is far more involved than the preservation of the precise sequence of bits constituting instrument reading, an ASCII text, or a bitmap graphic document.

All of the electronic records in lesser or greater degrees rely for access on the use of technologies that arise and evolve rapidly and just as rapidly become obsolete. Computing platforms on which the records are created, preserved, or examined, communication infrastructures interconnecting these platforms, data recording media, and, perhaps most importantly, data recording formats are all subject to rapid obsolescence while the records themselves must persist.

Preservation approaches for electronic records are multifold and can be broadly categorized in following areas of concern:

- Media
- Hardware Technology
- Software Technology, including record formats
- Archival: provenance, authenticity, context, structure and appearance.

A significant complicating factor in preservation of an electronic record is the necessity to preserve some of the associated linkages to other records. The loss of such linkages may, at best, lead to the loss of context or, at worst, render the record itself unreadable.

Preservation of electronic records is the end-to-end process which enables re-production of an authentic copy of the record. To assure that reproduction preservation of electronic records extends beyond protection of the record physical medium to protection of record accessibility and assuring record authenticity over time. Assuring persistence of records means ensuring that the records are not only readable but also intelligible after the passage of time. Assuring record authenticity means ensuring that the records are not inadvertently or deliberately altered or corrupted over time and that the authenticity can be adequately proven. [10]

Finally, a fundamentally important aspect of ensuring that the records are accessible over time is appropriate processing of the records as they enter the electronic archive with respect to establishing appropriate searchable archival structures and relationships and extraction and storage of associated metadata. Preservation methods which maintain dependencies of records to obsolete technologies tend to increasingly constrain access over time. Continuing general public access to old and obsolete technologies may not be possible except in highly limited environments or circumstances. In example, general public access to an emulator appropriate to enable reliable future rendition of electronic records created in the technology of an early 1990's proprietary geographic information system in the hypothetical context of 2020 vintage computing is not presently assured.

4. System Characteristics and Drivers

The ERA system, because of its size, scope, ingest and access loads, and commitment to long term preservation and servicing of government records, will require deployment and design approaches that support its unique nature and mission goals. When designing and deploying ERA, NARA must take a long term view for the system's operation and its required scalability, reliability, and cost effective operations. This long term vision will accommodate the use of outsourcing of potential processing and hosting services while at the same time ensures NARA's stewardship of the records trusted to it.

4.1 Design and Deployment Goals

There are certain assumptions and drivers that sculpt the deployment approach for ERA. These assumptions and design drivers are collectively considered the design and deployment goals for the ERA program. These goals include:

- NARA must own and control at least one set of all holdings of electronic records entrusted to it. This is required for protection of the records and fulfillment of NARA's mission to ensure long term preservation and access to the government's records.
- The ERA system is one of NARA's contributions to the Federal Enterprise Architecture (FEA) and fulfills a critical role in the development and deployment of NARA's own Enterprise Architecture (EA).
- The design and deployment vision for ERA must allow for the contracting out of record processing and access support, if NARA chooses to exercise that option in the future. The contracting out of record services must be done within the context of NARA's mission and ultimate responsibility for the integrity of the records.
- Minimize government ownership of equipment and facilities. This desire must be balanced against NARA's stewardship of the records and commitment to FEA support.
- Allow industry and academia to provide value added services on record holdings.
- Produce a highly reliable system design. Characteristics of such a design include:
    - Avoidance of single point/site of failure.
    - Graceful performance degradation of the system when failures occur
    - Maintain system operations in face of remedial maintenance (RM), preventative maintenance (PM), and planned upgrades/changes

4.2 System Design Drivers

In addition to the deployment goals, the design and deployment of ERA must take into account certain architectural demands and aspects of the ERA record preservation domain itself. These drivers must be accommodated in any deployment and design strategy for the ERA system. These drivers include:

- The size of the ERA record holdings. ERA permanent records holdings are projected to be in excess of 100 PBs of data 12-15 years after deployment, with continued growth in holdings in subsequent years. The sheer volume of data that must be accommodated, as well as its associated access loads, is a huge driver that must be accounted for in the ERA architecture. Architectural concepts including distributed deployment(s), load balancing techniques, and multiple sources for access to high demand records are applicable to the holdings size aspect of ERA.
- Insuring the integrity of the record holdings. The records must be protected from loss, alteration, or the lack of access capability over time. Appropriate security and accommodation of timely backup of holdings with subsequent restoration of access are techniques that are required in this area.
- The evolutionary nature of the ERA system. This aspect is most pronounced in two areas:
  - Changes to the Persistent Preservation approaches used for records. Over time electronic records will need to be stored, represented, and accessed in different ways given the forward march of computer technology and the rapid obsolescence of formats and techniques.
  - Independent of the record preservation techniques, the general infrastructure and support technologies used in ERA will need to be updated and upgraded over time. Technology insertion into the ERA design will be imperative.
- The heterogeneity of assets in ERA will complicate storing and providing access to the assets, as well as preserving them. The scope of this issue can be appreciated by considering that ERA records can be classified via three different attributes.
  - Record Types (RTs) – Any record will be classified according to its intellectual format. Examples of record types include letters, ledgers, maps, reports, etc.
  - Data types (DTs) – A data type is a set of lexical representations for a corresponding set of values. The values might be alphabetic characters, numbers, colors, shades of grey, sounds, et al. The lexical representation of such values in digital form assigns each value to a corresponding binary number, or string of bits. A data type may be simple, such as the ASCII representation of alphabetic characters, or composite; that is, consisting of a combination of other data types. An electronic record consists of one or more digital components; that is, strings of bits each of which has a specific data type.
  - Varied classes/collections of holdings – Records of the same RT and DT may still belong to different record series or collections, which further define the nature of the record. Examples of high level collections or

series could include particular Presidential collections, Federal record series, and potentially record series in Federal Record Centers (FRCs).

5. Conclusion

This paper has provided an overview of the NARA ERA program and the challenges that face NARA is the areas of electronic records preservation, system deployment, and archival management of the Nation's permanent records. The NARA ERA program represents a bold initiative in electronic records management and preservation and is a call for industry to propose new and innovative approaches to the unique issues NARA faces as the steward of the government's electronic records. Through the fusion of different technologies such as distributed computing, large scale object storage and access methods, secure infrastructure, and forward thinking record preservation strategies, ERA will open and exciting new era for electronics records management and access.

References:

[1] The Networking and Information Technology, R&D (NITRD), SUPPLEMENT TO THE PRESIDENT'S BUDGET FOR FY 2004, A Report by the Interagency Working Group on Information Technology Research and Development National Science and Technology Council, September 2003.

[2] The Networking and Information Technology, R&D (NITRD), SUPPLEMENT TO THE PRESIDENT'S BUDGET FOR FY 2003, A Report by the Interagency Working Group on Information Technology Research and Development National Science and Technology Council, July 2002

[3] The Networking and Information Technology, R&D (NITRD), SUPPLEMENT TO THE PRESIDENT'S BUDGET FOR FY 2002, A Report by the Interagency Working Group on Information Technology Research and Development National Science and Technology Council, July 2001

[4] Building an Electronic Records Archives and Records Administration: Recommendation for Initial Development, http://www.nap.edu/openbook/0309089476/html/R1.html

[5] http://www.archives.gov/about_us/reports/2002_annual_report_measuring_success.pdf

[6] NARA's Strategic Directions for Federal Records Management, July 31, 2003, http://www.archives.gov/records_management/pdf/strategic_directions.pdf, referenced 10/2003.

[7] *How Much Information? 2003,* http://www.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm#summary

[8] *Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years*, Kenneth Thibodeau, July 10, 2002

[9] *Preserving the Long-Term Authenticity of Electronic Records: The InterPARES Project*, Heather MacNeil, University of British Columbia AABC Newsletter, Volume 10 No. 2 Spring 2000

http://aabc.bc.ca/aabc/newsletter/10_2/preserving_the_long.htm
[10] The Long-Term Preservation of Authentic Electronic Records: Findings of the InterPares Project. 2003. http://www.interpares.org/book/index.cfm & http://www.archives.gov/electronic_records_archives/pdf/preservation_and_access_levels.pdf

Preservation Environments

Reagan W. Moore
San Diego Supercomputer Center
University of California, San Diego
9500 Gilman Drive, MC-0505
La Jolla, CA 92093-0505
moore@sdsc.edu
tel: +1-858-534-5073
fax: +1-858-534 5152

Abstract:
The long-term preservation of digital entities requires mechanisms to manage the authenticity of massive data collections that are written to archival storage systems. Preservation environments impose authenticity constraints and manage the evolution of the storage system technology by building infrastructure independent solutions. This seeming paradox, the need for large archives, while avoiding dependence upon vendor specific solutions, is resolved through use of data grid technology. Data grids provide the storage repository abstractions that make it possible to migrate collections between vendor specific products, while ensuring the authenticity of the archived data. Data grids provide the software infrastructure that interfaces vendor-specific storage archives to preservation environments.

1. Introduction

A preservation environment manages both archival content (the digital entities that are being archived), and archival context (the metadata that are used to assert authenticity) [8]. Preservation environments integrate data storage repositories with information repositories, and provide mechanisms to maintain consistency between the context and content. Preservation systems rely upon software systems to manage and interpret the data bits. Traditionally, a digital entity is retrieved from an archival storage system, structures within the digital entity are interpreted by an application that issues operating system I/O calls to read the bits, and semantic labels that assign meaning to the structures are organized in a database. This process requires multiple levels of software, from the archival storage system software, to the operating system on which the archive software is executed, to the application that interprets and displays the digital entity, to the database that manages the descriptive context. A preservation environment assumes that each level of the software hierarchy used to manage data and metadata will change over time, and provides mechanisms to manage the technology evolution.

A digital entity by itself requires interpretation. An archival context is needed to describe the provenance (origin), format, data model, and authenticity [9]. The context is created by archival processes, and managed through the creation of attributes that describe the knowledge needed to understand and display the digital entities. The archival context is organized as a collection that must also be preserved. Since archival storage systems manage files, software infrastructure is needed to map from the archival repository to the

preservation collection. Data Grids provide the mechanisms to manage collections that are preserved on vendor-supplied storage repositories [7].

Preservation environments manage collections for time periods that are much longer than the lifetime of any storage repository technology. In effect, the collection is held invariant while the underlying technology evolves. When dealing with Petabyte-sized collections, this is a non-trivial problem. The preservation environment must provide mechanisms to migrate collections onto new technology as it becomes available. The driving need behind the migrations is to take advantage of lower-cost storage repositories that provide higher capacity media, faster data transfer rates, smaller foot-print, and reduced operational maintenance. New technology can be more cost effective.

2. Persistent Archives and Data Grids

A persistent archive is an instance of a preservation environment [9]. Persistent archives provide the mechanisms to ensure that the hardware and software components can be upgraded over time, while maintaining the authenticity of the collection. When a digital entity in migrated to a new storage repository, the persistent archive guarantees the referential integrity between the archival context, and the new location of the digital entity. Authenticity also implies the ability to manage audit trails that record all operations performed upon the digital entity, access controls for asserting that only archivists performed the operations, and checksums to assert the digital entity has not been modified between applications of archival processes.

Data grids provide these data management functions in addition to abstraction mechanisms for providing infrastructure independence [7]. The abstractions are used to define the fundamental operations that are needed on storage repositories to support access and manipulation of data files. The data grid maps from the storage repository abstraction to the protocols required by a particular vendor product. By adding drivers for each new storage protocol as they are created, it is possible for a data grid to manage digital entities indefinitely into the future. Each time a storage repository becomes obsolete, the digital entities can be migrated onto a new storage repository. The migration is feasible as long as the data grid uses a logical name space to create global, persistent identifiers for the digital entities. The logical name space is managed as a collection, independently of the storage repositories. The data grid maps from the logical name space identifier to the file name within the vendor storage system.

Data grids support preservation by applying mappings to the logical name space to define the preservation context. The preservation context includes administrative attributes (location, ownership, size), descriptive attributes (provenance, discovery attributes), structural attributes (components within a compound record), and behavioral attributes (operations that can be performed on the digital entity). The context is managed as metadata in a database. An information repository abstraction is used to define the operations required to manipulate a collection within a database, providing the equivalent infrastructure independence mechanisms for the collection.

Archivists apply archival processes to convert digital entities into archival forms. Similar ideas of infrastructure independence can be used to characterize and manage archival processes. The application of each archival process generates part of the archival context. By creating an infrastructure independent characterization of the archival processes, it becomes possible to apply the archival processes in the future. An archival form can then consist of the original digital entity and the characterization of the archival process. Virtual data grids support the characterization of processes and on demand application of the process characterizations. A reference to the product generated by a process can result in direct access to the derived data product, or can result in the application of the process to create the derived data product. Virtual data grids can characterize and apply archival processes.

Data grids provide the software mechanisms needed to manage the evolution of software infrastructure [7] and automate the application of archival processes. The standard capabilities provided by data grids were assessed by the Persistent Archive Research Group of the Global Grid Forum [8]. Five major categories were identified that are provided by current data grids:

1. Logical name space; a persistent and infrastructure independent naming convention
2. Storage repository abstraction; the operations that are used to access and manage data
3. Information repository abstraction; the operations that are used to organize and manage a collection within a database
4. Distributed resilient architecture; the federated client-server architecture and latency management functions needed for bulk operations on distributed data
5. Virtual data grid; the ability to characterize the processing of digital entities, and apply the processing on demand.

The assessment compared the Storage Resource Broker (SRB) data grid from the San Diego Supercomputer Center [18], the European DataGrid replication environment (based upon GDMP, a project in common between the European DataGrid [2] and the Particle Physics Data Grid [15], and augmented with an additional product of the European DataGrid for storing and retrieving meta-data in relational databases called Spitfire and other components), the Scientific Data Management (SDM) data grid from Pacific Northwest Laboratory [20], the Globus toolkit [3], the Sequential Access using Metadata (SAM) data grid from Fermi National Accelerator Laboratory [19], the Magda data management system from Brookhaven National Laboratory [6], and the JASMine data grid from Jefferson National Laboratory [4]. These systems have evolved as the result of input by user communities for the management of data across heterogeneous, distributed storage resources.

EGP, SAM, Magda, and JASMine data grids support high energy physics data. The SDM system provides a digital library interface to archived data for PNL and manages data from multiple scientific disciplines. The Globus toolkit provides services that can be composed to create a data grid. The SRB data handling system is used in projects for

multiple US federal agencies, including the NASA Information Power Grid (digital library front end to archival storage) [11], the DOE Particle Physics Data Grid (collection-based data management) [15], the National Library of Medicine Visible Embryo project (distributed data collection) [21], the National Archives Records Administration (persistent archive research prototype) [10], the NSF National Partnership for Advanced Computational Infrastructure (distributed data collections for astronomy, earth systems science, and neuroscience) [13], the Joint Center for Structural Genomics (data grid) [5], and the National Institute of Health Biomedical Informatics Research Network (data grid) [1].

The systems therefore include not only data grids, but also distributed data collections, digital libraries and persistent archives. Since the core component of each system is a data grid, common capabilities do exist across the multiple implementations. The resulting core capabilities and functionality are listed in Table 1.

These capabilities should encompass the mechanisms needed to implement a persistent archive. This can be demonstrated by mapping the functionality required by archival processes onto the functionality provided by data grids.

Core Capabilities and Functionality
Storage repository abstraction
Storage interface to at least one repository
Standard data access mechanism
Standard data movement protocol support
Containers for data
Logical name space
Registration of files in logical name space
Retrieval by logical name
Logical name space structural independence from physical file
Persistent handle
Information repository abstraction
Collection owned data
Collection hierarchy for organizing logical name space
Standard metadata attributes (controlled vocabulary)
Attribute creation and deletion
Scalable metadata insertion
Access control lists for logical name space
Attributes for mapping from logical file name to physical file
Encoding format specification attributes
Data referenced by catalog query
Containers for metadata
Distributed resilient scalable architecture
Specification of system availability
Standard error messages
Status checking
Authentication mechanism
Specification of reliability against permanent data loss
Specification of mechanism to validate integrity of data
Specification of mechanism to assure integrity of data
Virtual Data Grid
Knowledge repositories for managing collection properties
Application of transformative migration for encoding format
Application of archival processes

Table 1. Core Capabilities of Data Grids

3. Persistent Archive Processes

The preservation community has identified standard processes that are applied in support of paper collections, listed in Table 2. These standard processes have a counterpart in the creation of archival forms for digital entities. The archival form consists of the original bits of the digital entity plus the archival context that describes the origin (provenance) of the data, the authenticity attributes, and the administrative attributes. A preservation environment applies the archival processes to each digital entity through use of a dataflow system, records the state information that results from each process, organizes the state information into a preservation collection, transforms the digital entity into a sustainable

format, archives the original digital entity and its transforms, and provides the ability to discover and retrieve a specified digital entity.

Archival Process	Functionality
Appraisal	Assessment of digital entities
Accession	Import of digital entities
Description	Assignment of provenance metadata
Arrangement	Logical organization of digital entities
Preservation	Storage in an archive
Access	Discovery and retrieval

Table 2. Archival process functionality for paper records

To understand whether data grids can meet the archival processing requirements for digital entities, scenarios are given below for the equivalent operations on digital entities. The term record is used to denote a digital entity that is the result of a formal process, and thus a candidate for preservation. The term fonds is used to denote a record series.

Appraisal is the process of determining the disposition of records and in particular which records need long-term preservation. Appraisal evaluates the various terms and conditions applying to the preservation of records beyond the time of their active life in relation to the affairs that created them. An archivist bases an appraisal decision on the uniqueness of the record collection being evaluated, its relationship to other institutional records, and its relationship to the activities, organization, functions, policies, and procedures of the institution.

Data grids provide the ability to register digital entities into a logical name space organized as a collection hierarchy for comparison with other records of the institution that have already been accessioned into the archives. The logical name space is decoupled from the underlying storage systems, making it possible to reference digital entities without moving them. The metadata associated with those other collections assist the archivist in assessing the relationship of the records being appraised to the prior records. Queries are made on the descriptive and provenance metadata to identify relevant records. The data grid supports controlled vocabularies for describing provenance and formats. This metadata also provides information that helps the archivist understand the relevance/importance/value of the records being appraised for documenting the activities, functions, etc. of the institution that created them. The activities of the institution can be managed as relationships maintained in a concept space, or as process characterizations maintained in a procedural ontology. By authorizing archivist access to the collection, and providing mechanisms to ensure authenticity of the previously archived records, the preservation environment maintains an authentic environment**.**

Accessioning is the formal acceptance into custody and recording of an acquisition. Data Grids control import by registering the digital entities into a logical name space organized

as a collection/sub-collection hierarchy. The records that are being accessioned can be managed as a collection independently of the final archival form. By having the data grid own the records (stored under a data grid Unix ID), all accesses to the records can be tracked through audit trails. By associating access controls with the logical name space, all references to the records can be authorized no matter where the records are finally stored.

Data grids put digital entities under management control, such that automated processing can be done across an entire collection. Bulk operations are used to move the digital entities using a standard protocol and to store the digital entities in a storage repository. Digital entities may be aggregated into containers (the equivalent of a cardboard box for paper) to control the data distribution within the storage repository. Containers are used to minimize the impact on the storage repository name space. The metadata catalog manages the mapping from the digital entities to the container in which they are written. The storage repository only sees the container names. Standard clients are used for controlling the bulk operations.

The information repository supports attribute creation and deletion to preserve record or fonds specific information. In particular, information on the properties of the records and fonds are needed for validation of the encoding formats and to check whether the entire record series has been received. The accession schedule may specify knowledge relationships that can be used to determine whether associated attribute values are consistent with implied knowledge about the collection, or represent anomalies and artifacts. An example of a knowledge relationship is the range of permissible values for a given attribute, or the expected number of records in a fonds. If the range of values do not match the assertions provided by the submitter, the archivist needs to note the discrepancy as a property of the collection.

Bulk operations are needed on metadata insertion when dealing with collections that contain millions of digital entities. A resilient architecture is needed to specify the storage system availability, check system status, authenticate access by the submitting institution, and specify reliability against data loss. At the time of accession, mechanisms such as checksums, need to be applied to be able to assert in the future that the data has not been changed.

The Open Archival Information System (OAIS) specifies submission information packages that associate provenance information with each digital entity [14]. While OAIS is presented in terms of packaging of information with each digital entity, the architecture allows bulk operations to be implemented. An example is bulk loading of multiple digital entities, in which the provenance information is aggregated into an XML file, while the digital entities are aggregated into a container. The XML file and container are moved over the network from the submitting site to the preservation environment, where they are unpacked into the storage and information repositories.

The integrity of the data (the consistency between the archival context and archival content) needs to be assured, typically by imposing constraints on metadata update.

When creating replicas and aggregating digital entities into containers, state information is required to describe the status of the changes. When digital entities are appended to a container, write locks are required to avoid over-writes. When a container is replicated, a synchronization flag is required to identify which container holds the new digital entities, and synchronization mechanisms are needed to update the replicas.

The accession process may also impose transformative migrations on encoding formats to assure the ability to read and display a digital entity in the future. The transformative migrations can be applied at the time of accession, or the transformation may be characterized such that it can be applied in the future when the digital entity is requested.

In order to verify properties of the entire collection, it may be necessary to read each digital entity, verify its content against an accession schedule, and summarize the properties of all of the digital entities within the record series. The summarization is equivalent to a bill of lading for moving the record series into the future. When the record series is examined at a future date, the archivist needs to be able to assert that the collection is complete as received, and that missing elements were never submitted to the archive. Summarization is an example of a collection property that is asserted about the entire record series. Other collection properties include completeness (references to records within the collection point to other records within the collection), and closure (operations on the records result in data products that can be displayed and manipulated with mechanisms provided by the archive). The closure property asserts that the archive can manipulate all encoding formats that are deposited into the archive.

Arrangement is the process and result of identification of documents for whether they belong to accumulations within a fonds or record series. Arrangement requires organization of both metadata (context) and digital entities (content). The logical name space is used as the coordination mechanism for associating the archival context with the submitted digital entities. All archival context is mapped as metadata attributes onto the logical name for each digital entity. The logical name space is also used as the underlying naming convention on which a collection hierarchy is imposed. Each level of the collection hierarchy may have a different archival context expressed as a different set of metadata. The metadata specifies relationships of the submitted records to other components of the record series. For a record series that has yearly extensions, a suitable collection hierarchy might be to organize each year's submission as a separate sub-collection, annotated with the accession policy for that year. The digital entities are sorted into containers for physical aggregation of similar entities. The expectation is that access to one digital entity will likely require access to a related digital entity. The sorting requires a specification of the properties of the record series that can be used for a similarity analysis. The container name in which a digital entity is placed is mapped as an administrative attribute onto the logical name. Thus by knowing the logical name of a digital entity within the preservation environment, all pertinent information can be retrieved or queried.

The process of arrangement points to the need for a digital archivist workbench. The storage area that is used for applying archival processes does not have to be the final

storage location. Data grids provide multiple mechanisms for arranging data, including soft-links between collections to associate a single physical copy with multiple sub-collections, copies that are separately listed in different sub-collections, and versions within a single sub-collection. Data grids provide multiple mechanisms for managing data movement, including copying data between storage repositories, moving data between storage repositories, and replicating data between storage repositories.

Description is the recording in a standardized form of information about the structure, function and content of records: Description requires a persistent naming convention and a characterization of the encoding format, as well as information used to assert authenticity. The description process generates the archival context that is associated with each digital entity. The archival context is includes not only the administrative metadata generated by the accession and arrangement processes, but also descriptive metadata that are used for subsequent discovery and access.

Preservation Function	Type of information
Administrative	Location, physical file name, size, creation time, update time, owner, location in a container, container name, container size, replication locations, replication times
Descriptive	Provenance, submitting institution, record series attributes, discovery attributes
Authenticity	Global Unique Identifier, checksum, access controls, audit trail, list of transformative migrations applied
Structural	Encoding format, components within digital entity
Behavioral	Viewing mechanisms, manipulation mechanisms

Table 3. Archival context managed for each digital entity

The description process can require access to the storage repository to apply templates for the extraction of descriptive metadata, as well as access to the information catalog to manage the preservation of the metadata. The description process should generate a persistent handle for the digital entity in addition to the logical name. The persistent handle is used to assert equivalence across preservation environments. An example of a persistent handle is the concatenation of the name of the preservation environment and the logical name of the entity, and is guaranteed unique as long as the preservation environments are uniquely named. The ability to associate a unique handle with a digital entity that is already stored requires the ability to apply a validation mechanism such as a digital signature or checksum to assert equivalence. If a transformative migration has occurred, the validation mechanism may require access to the original form of the digital entity.

Preservation is the process of protecting records of continuing usefulness: Preservation requires a mechanism to interact with multiple types of storage repositories, mechanisms for disaster recovery, and mechanisms for asserting authenticity.

The only assured mechanism for guaranteeing against content or context loss is the replication of both the digital entities and the archival metadata. The replication can implement bit-level equivalence for asserting that the copy is authentic. The replication must be done onto geographically remote storage and information repositories to protect against local disasters (fire, earthquake, flood). While data grids provide tools to replicate digital entities between sites, some form of federation mechanism is needed to replicate the archival context and logical name space. One would like to assert that a completely independent preservation environment can be accessed that replicates even the logical names of the digital entities. The independent systems are required to support recovery from operation errors, in which recovery is sought from the mis-application of the archival procedures themselves.

The coordination of logical name spaces between data grids is accomplished through peer-to-peer federation. Consistency controls on the synchronization of digital entities and metadata between the data grids are required for the user name space (who can access digital entities), the resources (whether the same repository stores data from multiple grids), the logical file names (whether replication is managed by the systems or archival processes), and the archival context (whether insertion of new entities is managed by the system or archival processes). Multiple versions of control policies can be implemented, ranging from automated replication into a union archive from multiple data grids, to simple cross-registration of selected sub-collections.

Data grids use a storage repository abstraction to manage interactions with heterogeneous storage systems. To avoid problems specific to vendor products, the archival replica should be made onto a different vendor's product from the primary storage system. The heterogeneous storage repositories can also represent different versions of storage systems and databases as they evolve over time. When a new infrastructure component is added to a persistent archive, both the old version and new version will be accessed simultaneously while the data and information content are migrated onto the new technology. Through use of replication, the migration can be done transparently to the users. For persistent archives, this includes the ability to migrate a collection from old database technology onto new database technology.

Persistence is provided by data grids through support for a consistent environment, which guarantees that the administrative attributes used to identify derived data products always remain consistent with migrations performed on the data entities. The consistent state is extended into a persistent state through management of the information encoding standards used to create platform independent representations of the context. The ability to migrate from an old representation of an information encoding standard to a new representation leads to persistent management of derived data products. It is worth noting that a transformative migration can be characterized as the set of operations performed on the encoding syntax. The operations can be applied on the original digital entity at the time of accession or at any point in the future. If a new encoding syntax standard emerges, the set of operations needed to map from the original encoding syntax to the new encoding syntax can be defined, without requiring any of the intermediate

encoding representations. The operations needed to perform a transformative migration are characterized as a digital ontology [8].

Authenticity is supported by data grids through the ability to track operations done on each digital entity. This capability can be used to track the provenance of digital entities, including the operations performed by archivists. Audit trails record the dates of all transactions and the names of the persons who performed the operations. Digital signatures and checksums are used to verify that between transformation events the digital entity has remained unchanged. The mechanisms used to accession records can be re-applied to validate the integrity of the digital entities between transformative migrations. Data grids also support versioning of digital entities, making it possible to store explicitly the multiple versions of a record that may be received. The version attribute can be mapped onto the logical name space as both a time-based snapshot of a changing record, and as an explicitly named version.

Access is the process of using descriptive metadata to search for archival objects of interest and retrieve them from their storage location. Access requires the ability to discover relevant documents, transport them from storage to the user, and interact with storage systems for document retrieval. The essential component of access is the ability to discover relevant files. In practice, data grids use four naming conventions to identify preserved content. A global unique identifier (GUID) identifies digital entities across preservation environments, the logical name space provides a persistent naming convention within the preservation environment, descriptive attributes support discovery based on attribute values, and the physical file name identifies the digital entity within a storage repository. In most cases, the user of the system will not know either the GUID, logical name or physical file name, and discovery is done on the descriptive attributes.

Access then depends upon the ability to instantiate a collection that can be queried to discover a relevant digital entity. A knowledge space is needed to define the semantic meaning of the descriptive attributes, and a mechanism is needed to create the database instance that holds the descriptive metadata. For a persistent archive, this is the ability to instantiate an archival collection from its infrastructure independent representation onto a current information repository. The information repository abstraction supports the operations needed to instantiate a metadata catalog.

The other half of access is transport of the discovered records. This includes support for moving data and metadata in bulk, while authenticating the user across administration domains. Since access mechanisms also evolve in time, mechanisms are needed to map from the storage and information repository abstractions to the access mechanism preferred by the user.

4. Preservation Infrastructure

The operations required to support archival processes can be organized by identifying which capability is used by each process. The resulting preservation infrastructure is shown in Table 4. The list includes the essential capabilities that simplify the management of collections of digital entities while the underlying technology evolves.

The use of each capability by one of the six archival processes is indicated by an x in the appropriate row. The columns are labeled by App (Appraisal), Acc (Accessioning), Arr (Arrangement), Des (Description), Pres (Preservation), and Ac (Access). Many of the data grid capabilities are required by all of the archival processes. This points out the difficulty in choosing an appropriate characterization for applying archival processes to digital entities. Even though we have shown that the original paper-oriented archival processes have a counterpart in preservation of digital entities, there may be a better choice for characterizing electronic archival processes.

Core Capabilities and Functionality	App	Acc	Arr	Des	Pres	Ac
Storage repository abstraction		x	x		x	x
Storage interface to at least one repository		x	x	x	x	x
Standard data access mechanism		x	x	x	x	x
Standard data movement protocol support		x	x	x	x	x
Containers for data		x	x		x	x
Logical name space	x	x	x	x	x	x
Registration of files in logical name space	x	x	x	x	x	
Retrieval by logical name		x	x		x	x
Logical name space structural independence from physical file	x	x	x	x	x	x
Persistent handle		x	x	x	x	x
Information repository abstraction	x	x	x	x	x	x
Collection owned data	x	x	x	x	x	x
Collection hierarchy for organizing logical name space	x	x	x	x		
Standard metadata attributes (controlled vocabulary)	x	x	x	x	x	x
Attribute creation and deletion	x	x	x	x	x	
Scalable metadata insertion		x	x	x	x	
Access control lists for logical name space	x	x	x	x	x	x
Attributes for mapping from logical file name to physical file		x	x		x	x
Encoding format specification attributes	x	x		x	x	x
Data referenced by catalog query						x
Containers for metadata		x	x	x	x	x
Distributed resilient scalable architecture	x	x	x	x	x	x
Specification of system availability		x			x	x
Standard error messages		x	x	x	x	x
Status checking		x	x	x	x	x
Authentication mechanism	x	x	x	x	x	x
Specification of reliability against permanent data loss	x	x	x	x	x	
Specification of mechanism to validate integrity of data		x	x	x	x	x
Specification of mechanism to assure integrity of data	x	x	x	x	x	x
Virtual Data Grid		x	x	x	x	x
Knowledge repositories for managing collection properties	x	x	x	x	x	x
Application of transformative migration for encoding format		x	x	x	x	x
Application of archival processes		x	x	x	x	x

Table 4. Data Grid capabilities used in preservation environments

5. Persistent Archive Prototype

The preservation of digital entities is being implemented at the San Diego Supercomputer Center (SDSC) through multiple projects that apply data grid technology. In collaboration with the United States National Archives and Records Administration (NARA), SDSC is developing a research prototype persistent archive. The preservation

environment is based on the Storage Resource Broker (SRB) data grid [17], and links three archives at NARA, the University of Maryland, and SDSC. For the National Science Foundation, SDSC has implemented a persistent archive for the National Science Digital Library [12]. Snapshots of digital entities that are registered into the NSDL repository as URLs are harvested from the web and stored into an archive using the SRB data grid. As the digital entities change over time, versions are tracked to ensure that an educator can find the desired version of a curricula module.

Both of these projects rely upon the ability to create archival objects from digital entities through the application of archival processes. We differentiate between the generation of archival objects through the application of archival processes, the management of archival objects using data grid technology, and the characterization of the archival processes themselves, so that archived material can be re-processed (or re-purposed) in the future using virtual data grids.

The San Diego Supercomputer Center Storage Resource Broker (SRB) is used to implement the persistent archives. The SRB provides mechanisms for all of the capabilities and functions listed in Table 2 except for knowledge repositories. The SRB also provides mechanisms for the extended features listed in section 3, such as soft-links, peer-to-peer federation of data grids, and mapping to user-preferred APIs. The SRB storage repository abstraction is based upon standard Unix file system operations, and supports drivers for accessing digital entities stored in Unix file systems (Solaris, SunOS, AIX, Irix, Unicos, Mac OS X, Linux), in Windows file systems (98, 2000, NT, XP, ME), in archival storage systems (HPSS, UniTree, DMF, ADSM, Castor, Dcache, Atlas Data Store), as binary large objects in databases (Oracle, DB2, Sybase, SQLServer, PostgresSGL), in object ring buffers, in storage resource managers, in FTP sites, in GridFTP sites, on tape drives managed by tape robots, etc. The SRB has been designed to facilitate the addition of new drivers for new types of storage systems. Traditional tape-based archives still remain the most cost-effective mechanism for storing massive amounts of data, although the cost of commodity-based disk is approaching that of tape [17]. The SRB supports direct access to tapes in tape robots.

The SRB information repository abstraction supports the manipulation of collections stored in databases. The manipulations include the ability to add user-defined metadata, import and export metadata as XML files, support bulk registration of digital entities, apply template-based parsing to extract metadata attribute values, and support queries across arbitrary metadata attributes. The SRB automatically generates the SQL that is required to respond to a query, allowing the user to specify queries by operations on attribute values.

Version 3.0.1 of the Storage Resource Broker data grid provides the basic mechanisms for federation of data grids [16]. The underlying data grid technology is in production use at SDSC and manages over 90 Terabytes of data comprising over 16 million files. The ultimate goal of the NARA research prototype persistent archive is to identify the key technologies that facilitate the creation of a preservation environment.

5. Summary

Persistent archives manage archival objects by providing infrastructure independent abstractions for interacting with both archival objects and software infrastructure. Data grids provide the abstraction mechanisms for managing evolution of storage and information repositories. Persistent archives use the abstractions to preserve the ability to manage, access and display archival objects while the underlying technologies evolve.

The challenge for the persistent archive community is the demonstration that data grid technology provides the correct set of abstractions for the management of software infrastructure. The Persistent Archive Research Group of the Global Grid Forum is exploring this issue, and is attempting to define the minimal set of capabilities that need to be provided by data grids to implement persistent archives [8]. A second challenge is the development of digital ontologies that characterize the structures present within digital entities. The Data Format Description Language research group of the Global Grid Forum is developing an XML-based description of the structures present within digital entities, as well as a description of the semantic labels that are applied to the structures. A third challenge is the specification of a standard set of operations that can be applied to the relationships within an archival object. A preservation environment will need to support operations at the remote storage repository, through the application of a digital ontology.

6. Acknowledgements

The concepts presented here were developed by members of the Data and Knowledge Systems group at the San Diego Supercomputer Center. The Storage Resource Broker was developed principally by Michael Wan and Arcot Rajasekar. This research was supported by the NSF NPACI ACI-9619020 (NARA supplement), the NSF NSDL/UCAR Subaward S02-36645, the DOE SciDAC/SDM DE-FC02-01ER25486 and DOE Particle Physics Data Grid, the NSF National Virtual Observatory, the NSF Grid Physics Network, and the NASA Information Power Grid. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation, the National Archives and Records Administration, or the U.S. government. This document is based upon an informational document submitted to the Global Grid Forum. The data grid and Globus toolkit characterizations were only possible through the support of the following persons: Igor Terekhov (Fermi National Accelerator Laboratory), Torre Wenaus (Brookhaven National Laboratory), Scott Studham (Pacific Northwest Laboratory), Chip Watson (Jefferson Laboratory), Heinz Stockinger and Peter Kunszt (CERN), Ann Chervenak (Information Sciences Institute, University of Southern California), Arcot Rajasekar (San Diego Supercomputer Center). Mark Conrad (NARA) provided the archival process characterization.

included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

7. References

1. Biomedical Informatics Research Network, http://nbirn.net/
2. EDG – European Data Grid, http://eu-datagrid.web.cern.ch/eu-datagrid/
3. Globus – The Globus Toolkit, http://www.globus.org/toolkit/
4. Jasmine – Jefferson Laboratory Asynchronous Storage Manager, http://cc.jlab.org/scicomp/JASMine/
5. Joint Center for Structural Genomics, http://www.jcsg.org/
6. Magda – Manager for distributed Grid-based Data, http://atlassw1.phy.bnl.gov/magda/info
7. Moore, R., C. Baru, "Virtualization Services for Data Grids", Book chapter in "Grid Computing: Making the Global Infrastructure a Reality", John Wiley & Sons Ltd, 2003.
8. Moore, R., A. Merzky, "Persistent Archive Concepts", Global Grid Forum Persistent Archive Research Group, Global Grid Forum 8, June 26, 2003.
9. Moore, R., "The San Diego Project: Persistent Objects," Archivi & Computer, Automazione E Beni Culturali, l'Archivio Storico Comunale di San Miniato, Pisa, Italy, February, 2003.
10. NARA Persistent Archive Prototype, http://www.sdsc.edu/NARA/Publications.html
11. NASA Information Power Grid (IPG) is a high-performance computing and data grid, http://www.ipg.nasa.gov/
12. National Science Digital Library, http://nsdl.org
13. NPACI National Partnership for Advanced Computational Infrastructure, http://www.npaci.edu/
14. OAIS - Reference Model for an Open Archival Information System (OAIS). submitted as ISO draft, http://www.ccsds.org/documents/pdf/CCSDS-650.0-R-1.pdf, 1999.
15. Particle Physics Data Grid, http://www.ppdg.net/
16. Peer-to-peer federation of data grids, http://www.npaci.edu/dice/srb/FedMcat.html
17. Rajasekar, A., M. Wan, R. Moore, G. Kremenek, T. Guptil, "Data Grids, Collections, and Grid Bricks", Proceedings of the 20th IEEE Symposium on Mass Storage Systems and Eleventh Goddard Conference on Mass Storage Systems and Technologies, San Diego, April 2003.
18. Rajasekar, A., M. Wan, R. Moore, "mySRB and SRB, Components of a Data Grid", 11th High Performance Distributed Computing conference, Edinburgh, Scotland, July 2002.
19. SAM – Sequential data Access using Metadata, http://d0db.fnal.gov/sam/.
20. SDM – Scientific Data Management in the Environmental Molecular Sciences Laboratory, http://www.computer.org/conferences/mss95/berard/berard.htm.
21. Visible Embryo Project, http://netlab.gmu.edu/visembryo.htm

Data Management as a Cluster Middleware Centerpiece

Jose Zero, David McNab, William Sawyer, Samson Cheung
Halcyon Systems, Inc.
1219 Folsom St
San Francisco CA 94103
Tel +1-415-255-8673, Fax +1-415-255-8673
e-mail: zero@halcyonsystems.com

Daniel Duffy
Computer Sciences Corporation
NASA NCCS, Goddard Space Flight Center
Greenbelt MD 20771
Tel +1-301-286-8830
e-mail: Daniel.Q.Duffy@gsfc.nasa.gov

Richard Rood, Phil Webster, Nancy Palm, Ellen Salmon, Tom Schardt
NASA NCCS, Goddard Space Flight Center
Greenbelt MD 20771
Tel: +1-301-614-6155, Fax: +1-301-286-1777
e-mail: Richard.B.Rood.1@gsfc.nasa.gov

Abstract
Through earth and space modeling and the ongoing launches of satellites to gather data, NASA has become one of the largest producers of data in the world. These large data sets necessitated the creation of a Data Management System (DMS) to assist both the users and the administrators of the data. Halcyon Systems Inc. was contracted by the NASA Center for Computational Sciences (NCCS) to produce a Data Management System. The prototype of the DMS was produced by Halcyon Systems Inc. (Halcyon) for the Global Modeling and Assimilation Office (GMAO). The system, which was implemented and deployed within a relatively short period of time, has proven to be highly reliable and deployable. Following the prototype deployment, Halcyon was contacted by the NCCS to produce a production DMS version for their user community. The system is composed of several existing open source or government-sponsored components such as the San Diego Supercomputer Center's (SDSC) Storage Resource Broker (SRB), the Distributed Oceanographic Data System (DODS), and other components. Since Data Management is one of the foremost problems in cluster computing, the final package not only extends its capabilities as a Data Management System, but also to a cluster management system. This Cluster/Data Management System (CDMS) can be envisioned as the integration of existing packages.

1. Introduction

In the last twelve years, Commercial Off-the-Shelf (COTS)-based cluster computing has become the main source of supercomputing providers. From the revolution of the first viable microprocessors that lead the way to replacing vector supercomputers, to passing through new network technologies and arriving at the efficient porting of scientific code, the road to cluster

computing was paved with problems seemingly impossible to resolve. In a sense, the battle was won; but the war is still being fought.

Many aspects of computing have changed so radically that situations from the past seem unbelievably irrelevant today. Up until 1999, computing centers spent an immense amount of time in lengthy negotiations with vendors in an effort to obtain "build-able operating system codes". Today, they can directly download them from the web.

Still, in the midst of a new era with the power of COTS microprocessors, there are many challenges. Despite networks with low latency and high bandwidth and build-able operating systems and the availability of a myriad of open source packages, cluster computing is, at best, a difficult task that fails to replace the panacea days of Cray Research Inc.'s delivery of a C90 supercomputer.

The Data Management System (DMS) attempts to fill the void of middleware that both supercomputing centers and their users need in order to easily manage and use the diverse technology of cluster computers. The DMS is composed of several existing open source or government-sponsored components, such as the San Diego Supercomputing Center's Storage Resource Broker (SRB), the Distributed Oceanographic Data System (DODS), and others. Since data management is one of the major concerns in High Performance Computing (HPC), the final DMS package not only serves as a data management system for very high end computing, but it can easily be extended to a complete cluster management system.

Many areas of science that base their results on computing resources have different ratios of Mega-Flops per byte of data ingested and/or produced. Meteorology is a science that ingests and produces voluminous amounts of data. It is not a coincidence that the same branch of science that produced the word "computer" is now leading the core issues of cluster computing.

One of the legacy items from the previous computing models of the 60's, 70's, 80's, and 90's is the separation of mass storage engines and computing clusters. At this point, it is more efficient to follow the management structure of the computing centers rather than the computing architecture of the systems. COTS mass storage units, with multiple terabytes of attached disks, are just as reliable and economical as the COTS computing nodes. COTS CPU power has grown side-by-side with high bandwidth internal interconnects and new devices like Serial ATA and others that can provide support for multi-terabyte storage on each single unit. At the same time, OS improvements (Linux, etc.) make it possible to support those large file systems.

In a generic scientific computing center, the problem that must be solved is how to manage the vast amount of data that is being produced by multiple users in a variety of formats. And, the added challenge is to do so in a manner that is consistent and that does not consume all of the users' time manipulating such data or all of the computer center's personnel in endless migrations from one system to another and from one accounting report to the next. This holds true across a broad range of actions from software engineering practices, to the production of code, to upgrading OS versions and patches, and includes changes in the systems, in accounting, in system engineering practices, and in the management of the actual scientific data.

Despite the best efforts of computing centers, "dead data" continues to mount up in mass storage vaults. The increasing cost of maintaining the storage, migrating, and in general curating can reach up to 40% of the total budget of a typical computing center. These curation activities (such as changing ownership, deleting, browsing, etc.) add to the burden of data management. Likewise, the proliferation of mass storage vaults is increasingly higher: two copies in situ, a third copy for catastrophic recovery, a copy in the computing engine (scratch) and additional copies wherever users need them (desktops, websites, etc.). This not only drives up costs, but it also undermines the collaboration among different scientists wherein data sharing becomes a limiting factor.

The cost and expertise necessary to deploy a Grid-useable computing node is too high for small computing groups. Groups of ten to twenty computer users typically have one or two system administrators and no system software developers, which makes the start-up cost beyond their reach (both in terms of dollars and expertise). As computing power increases, fewer groups need a true supercomputer platform. A successful Grid should easily deploy smaller nodes and maintain production level.

Finally, the lack of connection between the datasets and the software engineering practices (code version, patches, etc.) and the computing environment (CPU type, number of CPUs, etc.) limits the life of a dataset, its utility, and the scientific verification value.

In this paper we describe an integration effort composed of several existing packages that solves, to a large extent (but not totally), the data management problem for data coming out of a cluster computing environment. As a posteriori result we describe how the data management, essential to the utility of a cluster, becomes a centerpiece for its management. We also propose an ensemble set that can be used as a turn-key engine for a further integration of Cluster/Data Management into a full Grid/Data Management System ("Incoherent"). In this area, Halcyon proposes that Incoherent be an Open Source Project.

2. Basic Requirements for a Data Management System

The following list contains the basic requirements for the DMS.

- Ensure a single point of information wherein data is retrieved/searched. Though there might be many different interfaces, the initial point of contact for each interface should be the same.
- Provide system tools to cap storage costs and select datasets to be expunged.
- Provide methods for minimizing the number of data copies (and conceivably provide a live backup of the data). The copy that is more efficient to fetch should be the one that is accessed.
- Establish a linkage between data, scientific metadata, computing metadata, and configuration management data.
- Provide support for data migration whether it is from computing nodes to local storage (where users are) or from one storage system to another.
- Support plug and play of different visualization tools.

- Avoid multiple, full, or subset copies of datasets in the system by providing a Virtual Local Data capacity (data always feels local), along with the automatic use of local caches and sub-setting on-the-fly.
- Provide robust, easily deployed, grid-compatible security tools.
- Deploy with ease. Most department-type scientific groups do not have the resources to integrate a fully deployed Cluster Software Management and Mass Storage System.

3. Data Management System, Present Components

Halcyon Systems has integrated several packages to work together as a DMS:

- Storage Resource Broker (front-end mass storage, metadata catalog)
- Distributed Oceanographic Data System (transport layer, connection to manipulation and visualization tools)
- Configuration Management Software for all systems involved
- Distributed Oceanographic Data System and GrADS visualization tool

A minimal number of changes were implemented in the SRB software. A build tool and benchmarks were produced for ease of administration. Exit codes were changed to comply with standard UNIX command return codes. The underlying database is Oracle 9i running on Linux.

The DODS dispatch script is CGI-Perl. It was modified to make calls to SRB S-utilities to retrieve and cache files from SRB. Once a file has been transferred to local disk, it remains there until either the SRB version is modified or the cache fills and it is the oldest file. The DODS server authenticates as SRB identity "dods", and users who wish to export their files via DODS add read access for that user to the files' access control lists.

The DODS environment does not maintain a separate metadata catalog for managing semantic-based access to the data. There is presently no connection between DODS metadata, which is synthesized from the DODS-served file depending on the data format, and SRB metadata, which is stored in the MCAT associated with the file. MCAT data cannot yet be retrieved through DODS, nor is DODS-style synthesized metadata stored in MCAT.

Configuration Management Software is a set of commands enabling the user/administrator to enter changes in the specifically devoted tables created separately from the SRB tables in the Oracle database.

GrADS is already integrated with DODS; however, future work will have a separate server (GrADS-DODS server or GDS) fully integrated with SRB. In this way, a wider set of data manipulation and computation will be directly accessible to DMS users.

Note on GCMD integration: DMS uses SRB's "user defined metadata" facility to store GCMD-compliant metadata. We have defined site-standard user metadata attributes corresponding to the attributes defined in GCMD; then restricted their values based on GCMD convention. An application level tool replaces the general purpose SRB metadata manipulation client and enforces the conventions.

4. Existing Architecture

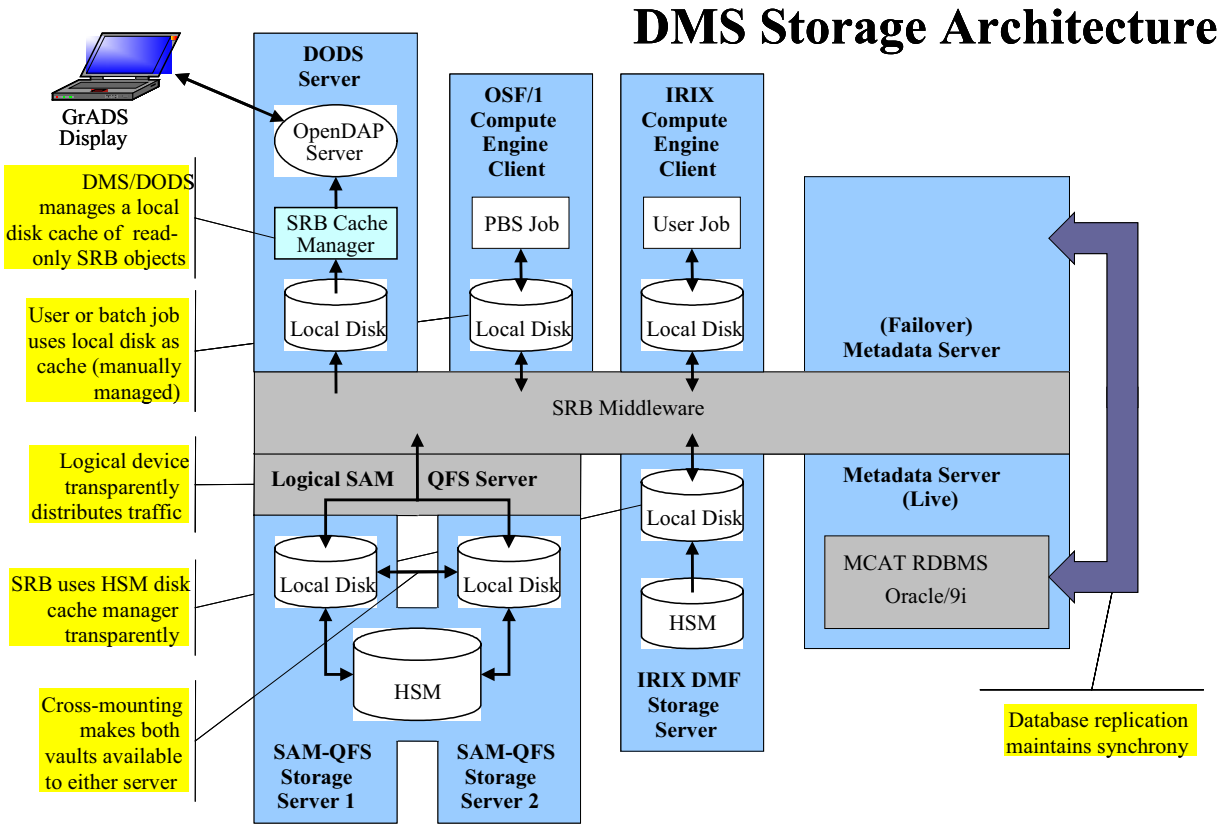


Figure 1: Depicts the existing components of the DMS deployed at the NCCS and their functionality.

5. Requirement Fulfillment

Based on the requirements and the architecture described above, the DMS currently meets the following requirements.

- There should be a single point of information for retrieving/searching the data. Even though there might be many different interfaces, the initial point of contact for each interface should be the same. SRB provides a single point of access for the data.

- The system should provide tools to cap storage costs and select datasets to be expunged. The Data Management System Toolkit provides tools to manage expiration dates for datasets and mechanisms allowing users to preserve selected datasets beyond a given lapse of time (separate description).

- The system should provide ways to minimize the number of data copies and could provide a live backup of the data. The copy that is more efficient to fetch should be the one fetched. DODS/OpenDAP can manage a local cache, network-wise close to the users. Computations, on-the-fly sub-setting can be provided by tools like the GrADS-DODs server (active implementation already on-going).

- Scratch space on the computing platforms can be managed by a short expiration date of a SRB replica of a given dataset.

- Linkage between data, scientific metadata, computing metadata. and configuration management data. SRB flexible metadata schemas provide a linkage between datasets and their scientific content. Metadata schema has been modified to accommodate the format provided by the Global Change Master Directory Software (GCMD), although a fully compatible version of GCMD has not been implemented as yet. Halcyon also has integrated a Configuration Management Software into the DMS that links the system "state" (patches, compilers, etc.) of computing engines with the dataset metadata.

- Provide support for data migration from computing nodes to local storage (where users are) or from one storage system to another: SRB provides bulk transfer from legacy mass storage systems to newer ones; and DODs/OpenDAP can manage local caches as datasets are requested by users. The Halcyon DMS Toolkit provides the following features:

    - file ownership management (user, group, project)
    - file expiration dates management tools
    - **dms acct** uses MCAT interface for accounting reports
    - **dms admin** provides administrative commands
    - **dms meta** provides metadata management, search
    - **dms ingest** stores files with metadata automatically
    - adds concept of file certification. A process through which the users can extend the life of a file beyond expiration dates.

- Provide robust, easily deployed, grid-compatible security tools. SRB's underlying security infrastructure is compatible with the Grid Security Infrastructure (GSI). At the moment, the current DMS deployment is using password encryption, which is more robust than FTP and does not pass clear text passwords. GSI can support tickets (PKI) and Kerberos infrastructure.

- Ease of deployment. Most department-type scientific groups do not have the resources to integrate a fully deployed Cluster Software Management and Mass Storage System. Halcyon is planning to deploy a turn-key server, named Infohedron, to deploy the DMS software in a single box (see next section).

6. Performance

As with all high performance production systems, the risk of *not* utilizing all available network bandwidth can be a significant issue. In tests performed between two single points at NCCS, the following results have proven that the DMS and, particularly, SRB are able to sustain performance levels equivalent to scp transfers without the overhead of CPU consumption due to encrypting and decrypting the data.

The NCCS implementation is built around a pair of redundant Linux-based SRB MCAT servers running Oracle/9i to provide database services. These DMS servers are identically configured two-CPU Xeon systems with 4 GBytes of RAM and SCSI RAID disk arrays. One machine, the primary, is the active server. The second is a hot backup that can be brought into production within two hours should a catastrophic failure disable the first, losing at most thirty minutes worth of MCAT transactions—although in the vast majority of situations the RAID arrays prevent this type of serious failure and no transactions will be lost.

DMS/SRB I/O bandwidth was measured between two hosts, "halem", a Compaq Tru64 compute cluster acting as SRB client, and "dirac", a Solaris9-based SAM-QFS storage server. The tests reported here used a single node of halem and a single node of dirac interconnected by Gigabit Ethernet. Thirty-two transfer threads ran simultaneously—although test results indicated that the performance changed little from eight to sixty-four nodes. These bandwidth tests were designed to demonstrate that DMS/SRB is capable of supporting the near-term projected storage load for NCCS, which was estimated at 2 TBytes per day with a ratio of three writes to one read—i.e., 1.5 TB write traffic and 0.5 TB read traffic per day. The average file at NCCS is 40 MBytes in size, and it was calculated that in order to meet the daily write requirement it would be necessary to complete the transfer of 1600 files in an hour. Although only one third this number of files had to be transferred within an hour to meet the read test requirements, for convenience the tests ran with the same group of 1600.

A significant part of the file transfer time is due to MCAT overhead independent of the file size, so the aggregate throughput increases significantly as the file size increases. For these tests, no NCCS-specific network optimization—for instance adjustment of network buffer sizes—took place.

TEST	ELAPSED m.	MB/s	TB/day
write	30.5 - 33.3	32 - 35	2.6 – 2.9
read	17.6 – 32.2	33 – 60	2.7 – 5.0

1600 40MB files, 32 threads, halem → dirac
requirement: 1 hr. or less, 2TB day (3:1 W:R)

NOTE: single client system to single server system;
no optimization to NCCS network

As the table demonstrates, DMS/SRB was easily able to meet the requirements even without optimization. The daily performance numbers were extrapolated from the 1600-file test performance.

The second group of tests measured MCAT transaction performance and were intended to demonstrate that DMS can support the expected number of file metadata operations per day. For the tests, it was estimated that each file would have 15 associated metadata attribute-value pairs, and similarly to the bandwidth tests a group of 1600 canonical 40 MByte files was used.

Metadata insertions and deletions were tested, as well as simple queries—display of the metadata attributes associated with a particular file. 50,000 insertions and deletions were required each day, as well as 10,000 searches.

DMS Performance: Metadata

TEST	ELAPSED m.	TRANS/s	TRANS/day
insert	43.5 – 48.6	8.2 – 9.2	711K – 795K
query	2.9 – 3.1	129 – 140	11.2M – 12.1M
delete	42.4 – 45.3	8.8 – 9.4	770K – 815K

1600 40MB files, 32 threads, halem → dirac
requirement: 50K inserts/day, 10K search/day

Even more so than with the bandwidth tests, the DMS/SRB easily exceeded the requirements.

7. Infohedron System Architecture

Presently, the DMS system is built on a Linux and Oracle 9i platform with limited redundancy (manual switchover), which covers the minimal needs of a production system. The cost of upgrading to a replicated database is largely due to the cost of an Oracle replicating database. Halcyon is testing the deployment of a Postgres-based, underlying database. In this area, Halcyon has been using an SRB 2.1 server while advancing to Postgres version 7.4. This decision has been based on the large customer base of Postgres – which allows it to mature faster – and the smaller customer base of SRB, which implies a slower maturation process of the software to arrive at the production level required by the NCCS environment. With an upgrade to SRB 3.0, the process would close the compatibility of Infohedron platforms by distributing the metadata catalog and, thereby, form a federated DMS.

In planning for the full deployment of Infohedron, Halcyon has included the GrADS-DODS server to fulfill the needs of NCCS major customers, such as the Global Modeling and Assimilation Office (GMAO), as well as the following packages (to make it useful to a wider audience of customers).

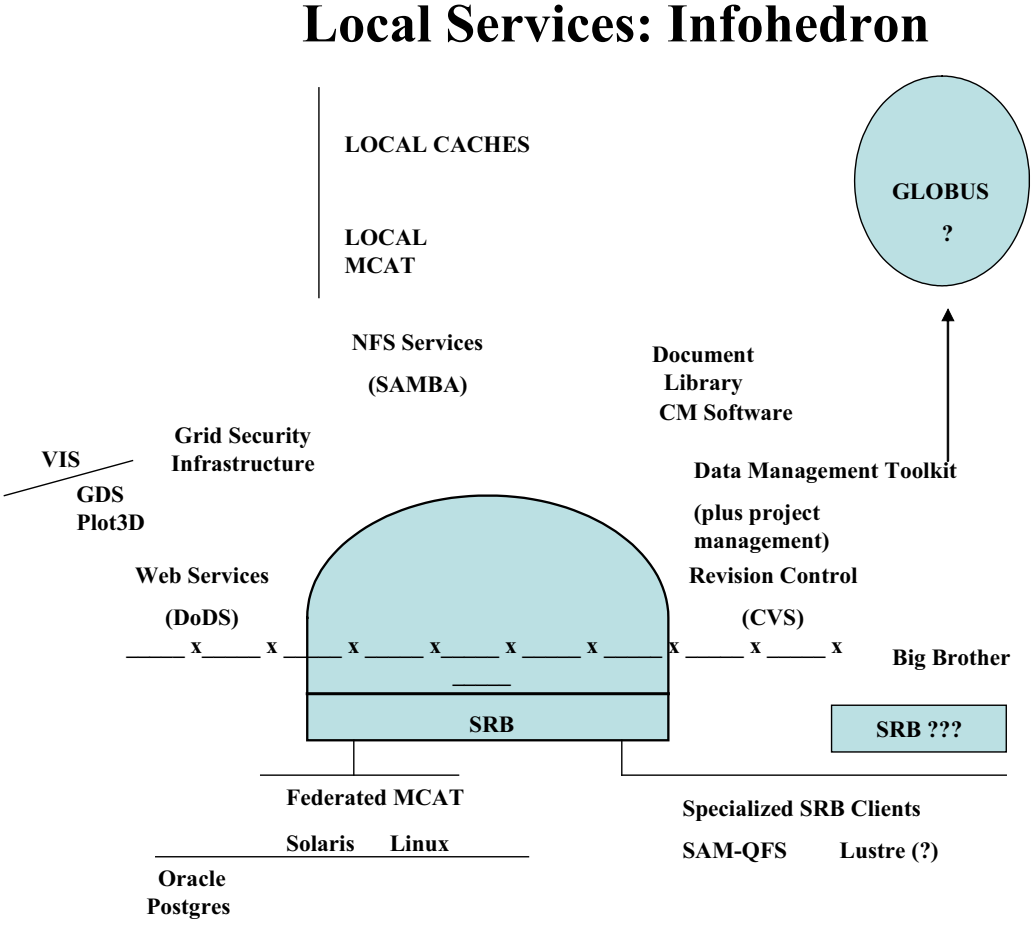


Figure 2: Depicts the turn-key option with typical services needed by a scientific group to adhere to a Grid-like infrastructure. The seemingly chaotic disposition of the packages is intended to depict large variations in needs from group-to-group. The question marks indicate uncertainties in the configuration of groups or the possibility of replacing them with other packages.

8. DMS as Cluster Management

By managing the accounting in the cluster and providing Virtual Locality for Data, DMS can provide full utilization of the cluster and the local caches co-located with the users and the scratch space of the computing cluster itself. By containing the software engineering information and the computing configuration management, DMS is able to provide data integrity and reproducibility.

Homogeneous, easily deployable security infrastructure couples with federated metadata catalogs enabling the Grid. Migration of data and underlying data movements can be controlled in a small environment automatically and in a larger environment with the aid of user indirect manipulation (SRB replication process). Finally user control of data sharing and user quotas (SRB 3.0) can enable cluster sharing, producing a CDMS.

9. Future Directions

Though many of the components described in this paper already exist, and their integration is relatively simple, the production level will be arduous to achieve. Halcyon provides a rigorous system engineering background to test, document and deploy all components. While the effort is sizeable, it has the potential to move progressively toward deployment of a large grid by doing the hardest work first – incorporating legacy data into a Data Management System and then enlarging the DMS into a wider set of services service like CDMS.

Parallel transfers of datasets over separate rails support is provided by SRB. However, it has not been tested under production on DMS.

GSI infrastructure has not been deployed at NCCS. The level of Software Systems support has not yet been determined.

Grid wise accounting has not yet been defined under CDMS.

The Earth System Modeling Framework (http://www.esmf.ucar.edu/) is in the process of formulating an I/O interface. The DMS project will provide a library to interact directly with DMS. If proper network support is provided, an application running in a computer cluster could directly deposit files into mass storage systems. In this way, a consolidation of high performance file-systems would provide savings, as well as avoid the usual double I/O process of depositing files in a local parallel file-system and then transporting them to mass storage.

Integration of the DMS with Lustre: Luster is a distributed file-system designed to provide high performance and excellent scalability for cluster computers. The resulting system would combine the simplicity, portability, and rich interfaces of DMS with the high performance and scalability of Lustre, effectively extending DMS to efficiently support data-intensive cluster-based supercomputing.

Lustre is designed to serve clusters with tens of thousands of nodes, manage petabytes of storage, and achieve bandwidths of hundreds of GBs/sec with state of the art security and management infrastructure. It is currently being developed with strong funding from the Department of Energy and corporate sponsors.

Experimentation with more integration between SRB and the underlying Hierarchical Storage Systems could lead to a more efficient sub-setting by extracting only necessary parts of the files to be sub-set directly from tape (no full file recalling). This is similar to the ECMWF MARS Archive.

In conclusion we propose a two-tier approach: Firstly, convert the typical mass storage/computing cluster architecture most computing centers have to a service rich Cluster/Data Management System Architecture as, for example, the one described in this paper. Secondly, produce a brick-like engine that can take care of most requirements of the diverse, medium- to small-size groups. These bricks would provide local data caches and direct connection to software trees, as well as many other services targeted to the individual groups.

In this manner local idiosyncrasies can be accommodated while maintaining a homogeneous systems engineering throughout a Computing Grid.

The further development of this project would be a breakthrough in data-intensive supercomputing, alleviating a persistent performance bottleneck by enabling efficient analysis and visualization of massive, distributed datasets. By exploiting dataset layout metadata to provide direct access to the relevant portions of the data, it is possible to avoid the performance limiting serialization traditionally imposed by requiring transfer of the entire dataset through a non-parallel mass storage system.

References

[1] Rajasekar, A., M. Wan, R. Moore, "mySRB and SRB, Components of a Data Grid", 11th High Performance Distributed Computing conference, Edinburgh, Scotland, July 2002.

[2] Arcot Rajasekar, Michael Wan, Reagan Moore, George Kremenek, Tom Guptil, "Data Grids, Collections, and Grid Bricks", Proceedings of the 20th IEEE Symposium on Mass Storage Systems and Eleventh Goddard Conference on Mass Storage Systems and Technologies, San Diego, April 2003.

[3] http://www.unidata.ucar.edu/packages/dods

[4] http://www.esmf.ucar.edu

[5] http://gcmd.gsfc.nasa.gov

[6] http://www.globus.org

[7] http://www.escience-grid.org.uk

[8] http://www.nas.nasa.gov/About/IPG/ipg.html

[9] http://www.globalgridforum.org

Regulating I/O Performance of Shared Storage with a Control Theoretical Approach

Han Deok Lee, Young Jin Nam, Kyong Jo Jung, Seok Gan Jung, Chanik Park
Department of Computer Science and Engineering / PIRL
Pohang University of Science and Technology
Kyungbuk, Republic of Korea
{cyber93,yjnam,braiden,javamaze,cipark}@postech.ac.kr
tel +82-54-279-5668
fax +82-54-279-5699

Abstract

Shared storage has become commonplace with recent trends in storage technologies, such as storage consolidation and virtualization, etc. Meanwhile, storage QoS, which guarantees different storage service requirements from various applications toward shared storage, is gaining in importance. This paper proposes a new scheme which combines a feedback-controlled leaky bucket with a fair queuing algorithm in order to deliver guaranteed storage service for applications competing for shared storage. It not only assures an agreed-upon response time for each application, but also maximizes the aggregate I/O throughput by proportionating unused bandwidth to other active applications. Simulation results under various types of competing I/O workloads validate the features of the proposed scheme.

1 Introduction

The explosive growth of on-line data in many applications, such as multimedia, e-business, ERP, etc., poses scalability and manageability problems with storage. The advent of storage consolidation through SAN and storage cluster has overcome the limitation of scalability in traditional directed-attached storage environments. Moreover, the introduction of a new abstraction layer between physical disks and storage management applications called storage virtualization reduces complexity in storage manageability dramatically. With these trends in storage technologies, a shared storage model is now accepted in many areas, such as storage service providers, departmental storage environments in an enterprise, etc.

In a shared storage environment, it is commonplace for different users or applications to share a physical disk resource. Moreover, each application assumes that the storage is owned by itself, implying that it demands to have a guaranteed storage service called storage QoS at all times no matter how many applications share the storage. The storage QoS can be specified in many aspects which include I/O performance, reliability/availability, capacity, cost, etc. The issue of delivering guaranteed I/O performance has been given a higher priority than the others [6, 7, 12]. In addition, Shenoy and Vin in [7] described how

partitioning storage bandwidth can satisfy the different I/O performance requirements from mixed types of applications.

Few disk scheduling algorithms exist with QoS in mind [6, 7]. YFQ [6] is an approximated version of Generalized Processor Sharing (GPS) [2] that allows each application to reserve a fixed proportion of disk bandwidth. However, when the I/O workload from an application becomes heavier, it cannot bound the maximum response time. Cello framework [7] schedules I/O requests from heterogeneous types of clients including real-time and best-effort applications. The drawback of the Cello framework is that it assumes the existence of an accurate device model.

This paper proposes a new scheme which combines a feedback-controlled leaky bucket with a fair queuing algorithm in order to deliver guaranteed storage service for applications competing for shared storage. It not only assures an agreed-upon response time for each application, but also maximizes the aggregate I/O throughput by proportionating the unused bandwidth to other active applications. The feedback-controlled leaky bucket at the front-end dynamically regulates I/O rates from each application. The fair queuing algorithm at the back-end partitions a disk bandwidth among multiple I/O workloads from different applications in a proportional manner. As a result, the proposed algorithm is expected to assure a demanded response time as well as to maximize storage utilization. The remainder of this paper is organized as follows. Section 2 describes basic assumptions and definitions for the proposed scheme. Section 3 gives a detailed description on the proposed scheme. Performance evaluations via simulation are given in Section 4. Finally, this paper concludes with Section 5.

2 Preliminaries

Assumptions and Definitions: We begin by providing a set of assumptions and definitions to be used throughout this paper for clear descriptions. First, we assume that the characteristics of an I/O workload featured by an average IOPS and an average request size are known. Second, I/O requests access the underlying storage randomly. We denote the underlying shared storage with S. Next, it is shared by a set of I/O workloads denoted with $WS = \{W_1, W_2, ..., W_n\}$. An I/O workload W_i demands I/O performance level of $\{iops_i, size_i, rt_i\}$ for the shared storage S, where $size_i$ is an average request size, $iops_i$ is an I/O arrival rate per second (briefly IOPS), and rt_i is a demanded response time with $iops_i$. Given I/O requests of size $size_i$, the response time of any I/O request is required not to exceed rt_i, unless the current arrival I/O rate from W_i is faster than $iops_i$. Given the maximum IOPS of the storage denoted with $IOPS_{max}$, we assume that it can provide $0.75 \times IOPS_{max}$ in a sustained manner. Denote with $IOPS_d$ the sustained maximum IOPS.

Simple Admission Control: Next, we describe how to systematically map demanded I/o performance from an I/O workload[1] and the underlying storage, called admission control. Given $WS = \{W_1, W_2, ..., W_n\}$ where W_i requires performance of $\{iops_i, size_i, rt_i\}$, the

[1]Hereafter, we interchangeably use an application and an I/O workload.

following procedure decides if or not underlying storage can guarantee the required different types of performance from WS. Figure 1 depicts this procedure graphically. In Section 4, we will show how to seek those $IOPS^T$ and RT^T values for two sets of I/O workloads based on this procedure.

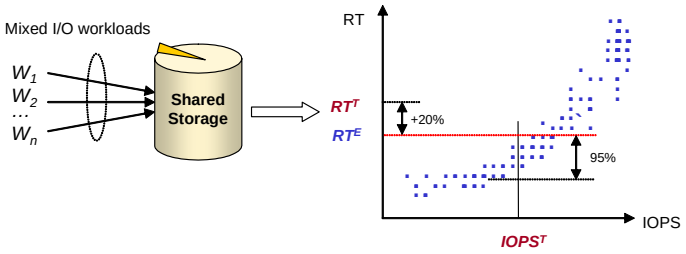


Figure 1: Measuring deliverable target response time (RT^T) for a given target IOPS ($IOPS^T$) with a set of I/O workloads WS

- generate mixed I/O requests whose size is $size_i$ with a probability of $iops_i/IOPS^T$, where $IOPS^T = \sum_{i=1}^{n} iops_i$,
- find a response time RT^E which is the 95th percentile of all response times whose corresponding IOPS falls into $IOPS^T$,
- compute a target response time with a 20% margin as follows: $RT^T = 120\%$ of RT^E, and
- if $rt_i \leq RT^T$ for all i, then it can be said that the underlying storage can guarantee the performance requirements demanded from WS.

3 The Proposed Algorithm

3.1 Feedback-Controlled Leaky Bucket (FCLB)

The proposed algorithm consists of a feedback-controlled leaky bucket and YFQ disk scheduling algorithm, as shown in Figure 2. The YFQ disk scheduling algorithm proportionately partitions a disk bandwidth according to the assigned weight (ϕ) among multiple I/O workloads, and then the feedback-controlled leaky bucket dynamically regulates requests within each partition by controlling the token replenish rate (ρ_i). The feedback-control module is composed of a monitor and controller. It controls the token replenish rate (ρ_i) parameter of the leaky bucket adaptively according to the current response time (RT_i). The controller increases ρ_i when RT_i goes below its demanded response time (rt_i). Conversely, the controller decreases ρ_i to its demanded IOPS $(iops_i)$ maximumly. In addition, when one I/O workload is inactive, the other can utilize the surplus left by the currently inactive I/O workload.

Monitor: The monitor component is responsible for collecting the current response time (RT_i) of each workload (W_i) at the end of each monitoring period, and feeding these results to the controller.

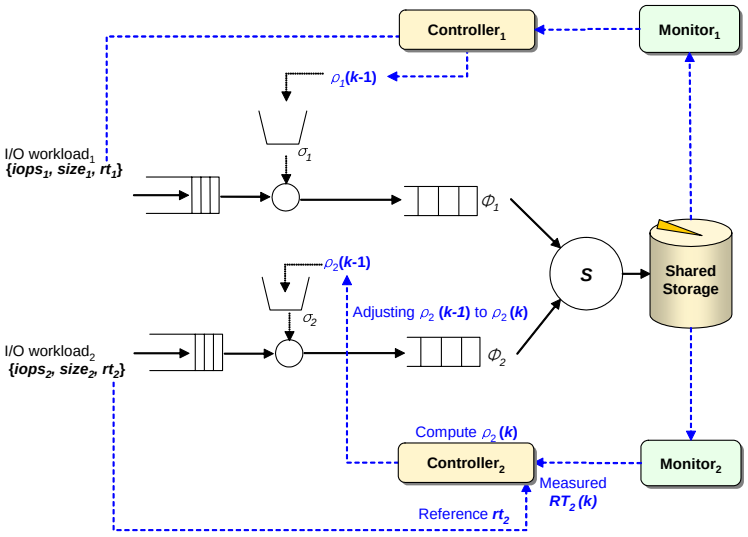


Figure 2: Architecture of the proposed algorithm

Controller: The controller compares the current response time $RT_i(k)$ for the time window $((k-1)I, kI)$ with the demanded response time (rt_i), and computes the replenish rate $\rho_i(k)$ to be used during the next monitoring period $(kI, (k+1)I)$.

1. For each workload i ($0 \leq i <$n) in the system, compute its error

$$E_i(k) \quad = \quad rt_i - RT_i(k), 0 \leq i < n \tag{1}$$

   where rt_i is called the reference in control theory. More negative values of $E_i(k)$ represent larger response time violations.

2. Compute the replenish rate according to the integral control function (K is a configurable parameter of the controller):

$$\rho_i(k) \quad = \quad \rho_i(k-1) + KE_i(k) \tag{2}$$

3. Adjust the replenish rate $\rho_i(k-1)$ in the previous control period to $\rho_i(k)$.

3.2 Feedback control loop in FCLB

Parameter K must to be tuned to prevent the replenish rate and measured response time from oscillating excessively and for fast convergence of the output to the reference. This can be done systematically with standard control theory techniques.

System Modeling: In general cases, all systems are non-linear. However, there are equilibrium points where systems behave in a linear fashion. Accordingly, non-linear systems can be linearized at the points previously described in Section 2. We approximate the controlled system with the linear model, as shown in Equation 3. The controlled system includes the shared storage, leaky bucket, monitor and controller. The output is $RT_i(k+1)$

and the input to the controlled system is the replenish rate $\rho_i(k)$ in the monitoring period $(kI, (k+1)I)$.

$$RT_i(k+1) - RT_i(k) \quad = \quad G \times (\rho_i(k) - \rho_i(k-1)) \tag{3}$$

The process gain, G, is the derivative of the output $RT_i(k+1)$ with respect to the input $\rho_i(k)$. G represents the sensitivity of the response time with regard to the change in the replenish rate.

z-Transform: Next, we transform the controlled system model to the z-domain, which is amenable to control analysis. The controlled system model in Equations 2 and 3 is equivalent to Equations 4 and 5. Figure 3 describes the flow of signals in the control loop.

$$\rho_i(z) \quad = \quad \frac{z}{z-1} K E_i(z) \tag{4}$$

$$RT_i(z) \quad = \quad \frac{1}{z} G \rho_i(z) \tag{5}$$

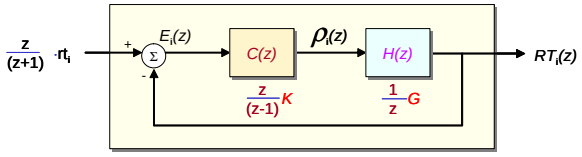


Figure 3: z-Transform of the control loop

Transfer Function: The whole feedback control system is modeled in the following transfer function:

$$\begin{aligned} H_c(z) \quad &= \quad \frac{C(z)H(z)}{1 + C(z)H(z)} \\ &= \quad \frac{KG}{z - (1 - KG)} \end{aligned} \tag{6}$$

Given the dynamic model of the closed loop system, we tune the control parameter K analytically using linear control theory, which states that the performance of a system depends on the poles of its closed loop transfer function. The closed loop transfer function has a single pole:

$$p \quad = \quad 1 - KG \tag{7}$$

and the sufficient and necessary condition for system stability is:

$$|p| < 1 \quad \Longleftrightarrow \quad 0 < K < \frac{2}{G} \tag{8}$$

4 Performance Evaluations

This section provides the behavior and performance of the proposed algorithm which is obtained from simulations. First, we describe simulation environments, I/O characteristics of two competing I/O workloads and performance requirements each of I/O workloads requires. Second, we will measure the I/O characteristics of the shared storage used in the experiments and investigate the range where shared storage can provide service in a stable manner. Third, given a set of competing I/O workloads and their performance requirements, we will perform an admission control to decide whether or not the underlying shared storage can assure the requirement. Fourth, we will determine experimentally two parameters G and K for the feedback control in order to stabilize the system. Finally, under a variety of conditions of the two I/O workloads, we will analyze the behavior and performance of the proposed algorithm.

4.1 Simulation Environments

We implemented the proposed algorithm within the DiskSim simulator[14]. Table1 shows the generic throttling parameters which are used for the experiments. In this table, ρ_i represents a rate of replenishing tokens. It is the same as the demanded maximum IOPS for W_i. σ_i means an amount of tokens that can be accumulated during an idle period. It corresponds to the size of a bucket in a leaky bucket model. Actually, tokens (IOPS) of $\frac{\rho_i}{\gamma}$ are replenished every $\frac{1}{\gamma}$ second. In our experiments, we will employ a time interval of 1 msec to replenish tokens to eventually allow I/O requests to pass through a throttling module and 1000 msec to control the replenish rate. We also set YFQ weight to ϕ_1:ϕ_2=2:1. Two competing I/O workloads based on a closed model are synthetically generated, as shown in Table2. The sizes of the I/O requests are distributed normally with a mean of 8 blocks. The performance of read and write is the same in a random I/O pattern, so that we will perform experiments with only read requests. We use a single IBM DNES309170W SCSI disk which serves arriving I/O requests in a FIFO manner.

Table 1: Throttling parameters for each experiment

Parameter	W_1	W_2
ρ_i ($\frac{iops_i}{\gamma}$)	$\frac{40}{\gamma}$	$\frac{20}{\gamma}$
Bucket Size (σ_i)	8	4

4.2 I/O characteristics of the shared storage

In this subsection we investigate the I/O characteristics of shared storage to decide a set of the replenish rate (ρ_i) where shared storage can provide service stably. The stable area is spread over 75 IOPS, as shown in Figure4. The growth of response times is gradual according to the increase of IOPS in this area. We respectively assign 40 and 20 IOPS to each I/O workload.

Table 2: Two competing I/O workloads

Parameter	W_1	W_2
$size_i$	4KB	4KB
$iops_i$	40	20
rt_i	35 msec	38 msec
access pattern	Random	Random

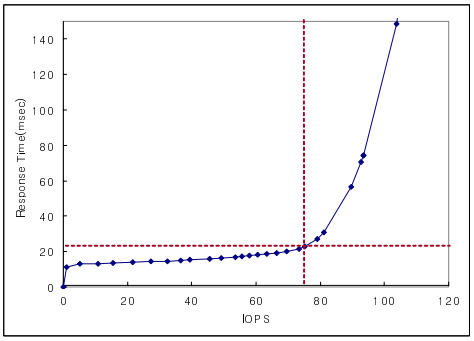


Figure 4: I/O characteristics of the shared storage with Random Read 4KB

4.3 Admission Control

After acquiring all the information about performance requirements from I/O workloads and underlying storage, we will try to map each I/O workload described in Table2 to the underlying storage. Recall the mapping methodology proposed in Section 2.

I/O requests of 4KB are issued increasingly to the corresponding reservation queue for $\frac{iops_i}{IOPS^T}$. We obtain IOPS versus RT chart as shown in Figure5. By analyzing Figure5 with several steps given in Section 2, we can obtain the following parameter in Table3. Based on $IOPS^T$ and RT^T of each I/O workload in Table3, it can be said that the demanded level of performance by given I/O workloads in Table2 can be deliverable with the underlying storage. RT^E is internally used for the feedback control as a reference value.

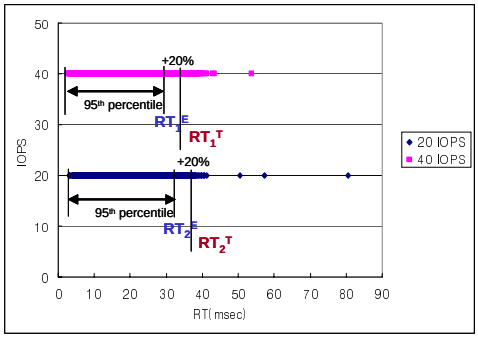
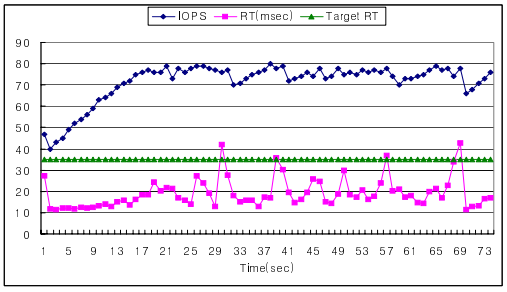


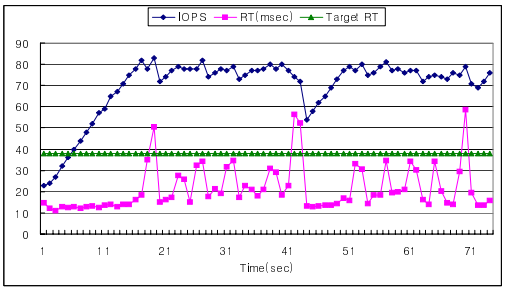
Figure 5: Measuring deliverable target response time (RT^T) for a given target IOPS $(IOPS^T)$ with two I/O workloads

Table 3: Given I/O workloads and deliverable response time by underlying storage

Parameter	$IOPS^T$	RT^E	RT^T
W_1	40	29.08 msec	34 msec
W_2	20	31.38 msec	37 msec



(a) Throughput and response time for Active 40 IOPS



(b) Throughput and response time for Active 20 IOPS

Figure 6: Performance Experiment 1. - one I/O workload is inactive and the other is active

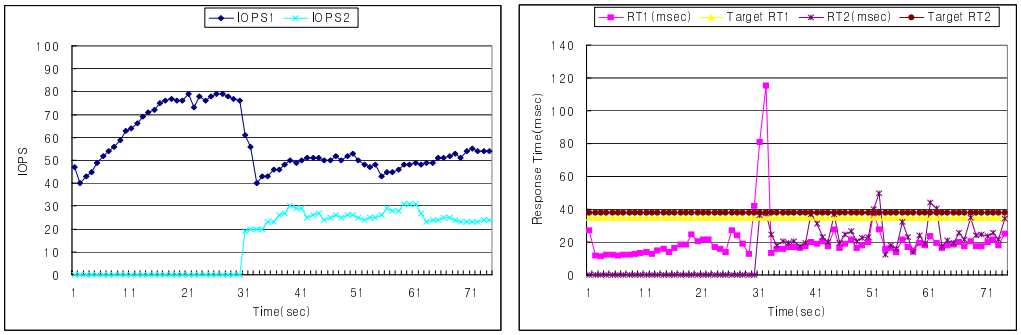
4.4 Control Parameters G, K

Here, we determine the parameters G and K. First, we approximate G by running a set of shared storage profiling experiments, as shown in Section 4.2. We estimate that G=0.3 for the synthetic workload. Since the closed loop transfer function as shown in Equation 7 has a single pole $p = 1 - KG$, we can set p to the desired value by choosing the right value of K. In our experiments, we set $p = +0.9$ by choosing $K = \frac{(1-p)}{G} = 0.33$. In the case of shared storage having a non-linear property whose I/O request service time is not proportional to its data size, we determine that the location of the pole is a close to $+1$ in order to stabilize the system.

4.5 Performance Results

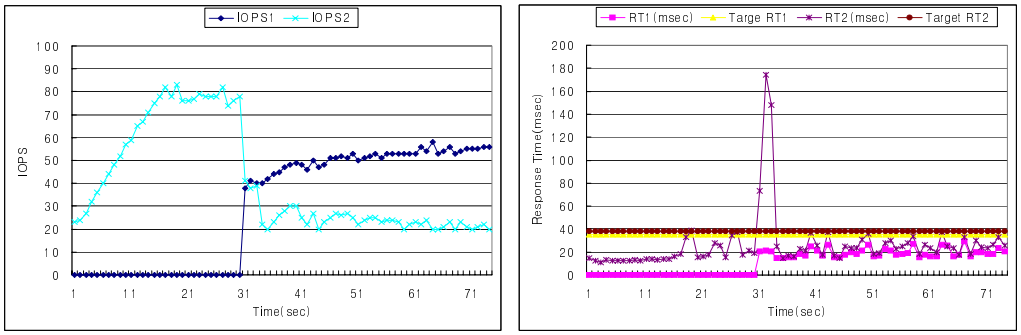
Under a variety of conditions of the two I/O workloads, we analyze the behavior of the proposed algorithm and its resulting I/O performance.

Case 1 - One Inactive I/O workload : In this experiment, one I/O workload is inactive and the other is active. Figure6(a)-(b) show time-plots of response time and throughput for two I/O workloads when 20 and 40 IOPS I/O workload are inactive respectively. As the graphs show, active I/O workload fully utilizes the shared storage by 72 IOPS on average. The response time time-plot shows that active I/O workload receives its demanded response time with a 5% violation. The degree of a response time violation seen by Figure6(b) is higher than Figure6(a). This is because the reference of 20 IOPS workload which is used to compute the error is larger than 40 IOPS.



(a) Throughput - 20 IOPS Step

(b) Response time - 20 IOPS Step



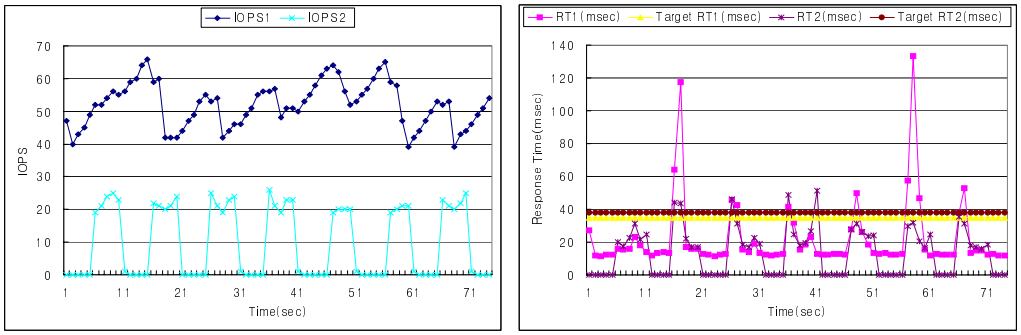
(c) Throughput - 40 IOPS Step

(d) Response time - 40 IOPS Step

Figure 7: Performance Experiment 2. - one I/O workload begins after 30 seconds and the other issues I/O requests continuously

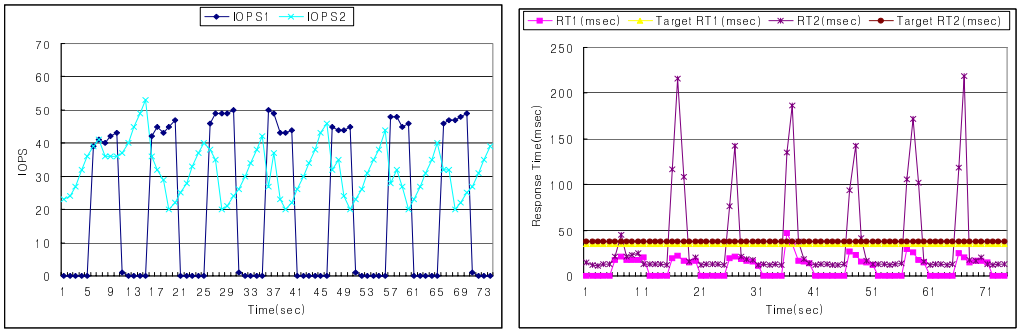
Case 2 - One Step I/O workload : In this experiment, one I/O workload begins after 30 seconds and the other issues I/O requests continuously. Figure7(a)-(b) shows the measured response time and throughput for 20 and 40 IOPS when 20 and 40 IOPS I/O workload begins after 30 seconds, respectively. In Figure7, we observe that two competing I/O workloads have its demanded response time in most cases except for 30 seconds where one I/O workload issues I/O requests and achieves its demanded IOPS in all cases. Before 30 seconds YFQ allocates a full disk bandwidth for continuously issuing the I/O workload. When one I/O workload comes on after 30 seconds, YFQ proportionately partitions a disk bandwidth according to the assigned weight among I/O workloads. As a result, the I/O workload allotted a full disk bandwidth has a high response time because it takes time for its reservation queue to drain sufficiently so that the corresponding response time target can be met. In this case, the response time violation is below 3 percent.

Case 3 - One Pulse I/O workload : In this experiment, one I/O workload repeats on for 5 seconds and off for 5 seconds and the other issues I/O requests continuously. As the graphs show, we can observe that two competing I/O workloads have its demanded IOPS in all cases. However, there is a spike in the response time whenever a burst of requests



(a) Throughput - 20 IOPS Pulse

(b) Response time - 20 IOPS Pulse



(c) Throughput - 40 IOPS Pulse

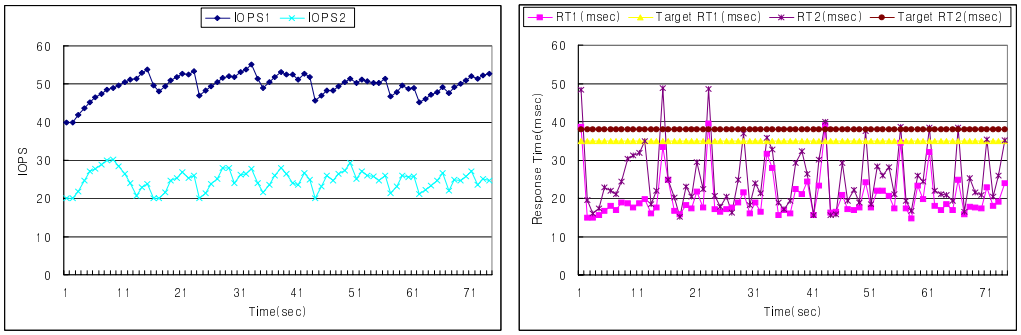
(d) Response time - 40 IOPS Pulse

Figure 8: Performance Experiment 3. - one I/O workload repeats on for 5 seconds and off for 5 seconds and the other issues I/O requests continuously

begins. The spike subsides quickly as soon as possible - within a two or three time window. This tendency is due to a feature of YFQ explained in our previous experiment. The degree of a response time violation seen by Figure8(b) is higher than Figure8(a). Also, this is due to the same reason described in our first experiment. The response time violation is 12/6%, 3/19%, as shown in Figure8(a)-(b).

Case 4 - Two Active I/O workload : In Figure9, two Active I/O workloads have its demanded IOPS and response time in most cases. In this case, the response time violation is below 3% and two I/O workloads occur at approximately the same rate as YFQ weight, that is 2:1.

Comparisons with Cello Framework [7]: The Cello framework heavily depends on the accuracy of the underlying storage device model, whereas the proposed scheme operates based on the measured performance of the underlying storage device. Thus, the proposed scheme can be more portable and applicable. In addition, the Cello framework proportionates unused storage performance by selecting pending I/O requests from the active applications in an ad-hoc order. However, the proposed scheme distributes the unused storage



(a) Throughput (b) Response time

Figure 9: Performance Experiment 4. - Two Active I/O workload

performance by adaptively configuring the replenishing rate of tokens at the leaky bucket of each application based on the concrete theory given in Equations 1–8.

5 Conclusion and Future Work

We proposed a new scheme that combines a feedback-controlled leaky bucket with a fair queuing algorithm in order to provide guaranteed storage service for different applications competing for shared storage. The proposed scheme not only assures an agreed-upon response time for each application, but also maximizes the aggregate I/O throughput by distributing the unused bandwidth to other active applications proportionally. We evaluated the performance of the proposed scheme under various types of competing I/O workloads. First, when an I/O workload becomes idle, we observed that the other workload could fully utilize the surplus bandwidth unused and only 5% of all completed I/O requests missed the agreed-upon response time. Second, when an I/O workload is backlogged again while the other I/O workload is using the entire bandwidth, we observed that competing I/O workloads had their demanded response time in most cases except for 30 seconds where an I/O workload issues I/O requests and achieved its demanded bandwidth in all cases. In this case, the response time violation is below 3 percent. Third, when an I/O workload is backlogged for a short period like a pulse while the other I/O workload is using the entire bandwidth, a spike occurs in the response time whenever a burst of requests begins. In this case, the proposed scheme revealed a lower performance than others. Finally, when both I/O workloads are active, both I/O workloads can approximately share the bandwidth at a rate of 2:1 and below 3% of all completed I/O requests missed the agreed-upon response time. In summary, the simulation results with various types of competing I/O workloads showed that the proposed algorithm provided a satisfactory level of response times; that is, 6% violation on average for the demanded response times. In future work, we plan to support workloads with multiple performance requirements that change over time.

6 Acknowledgement

The authors would like to thank the Ministry of Education of Korea for its support toward the Electrical and Computer Engineering Division at POSTECH through its BK21 program. This research was also supported in part by grant No. R01-2003-000-10739-0 from the Basic Research Program of the Korea Science and Engineering Foundation.

References

[1] J.S. Turner. *New directions in communications, or Which way to the information age?*. IEEE Communication Magazine, 1986.

[2] A. Parekh and R. Gallager. *A generalized processor sharing approach to flow control in integrated services networks: The single-node case*. IEEE/ACM Trans. on Networking, vol. 1, 1993.

[3] Pawan Goyal and Harrick M. Vin and Haichen Cheng. *Start-time Fair Queueing: A scheduling algorithms for integrated services packet switching networks*. Proceedings of SIGCOMM, 1996.

[4] E. Borowsky and R. Golding and A. Merchant and L. Schrier and E. Shriver and M. Spasojevic and J. Wilkes. *Using attribute-managed storage to achieve QoS*. Proceeding of 5th Intl. Workshop on Quality of Service, 1997.

[5] E. Borowsky and R. Golding and P. Jacobson and A. Merchant and L. Schreier and M. Spasojevic and J. Wilkes. *Capacity planning with phased workload*. Proceeding of First Intl. Workshop on Software and Performance, 1998.

[6] John L. Bruno and Jose Carlos Brustoloni and Eran Gabber and Banu Ozden and Abraham Silberschatz. *Disk Scheduling with quality of service guarantees*. Proceedings of the IEEE International Conference on Multimedia Computing and Systems, 1999.

[7] Shenoy, P., Vin, H.: *Cello: A disk scheduling framework for next-generation operating systems*. In: Proceedings of ACM SIGMETRICS. (1998)

[8] Guillermo A. Alvarez and Elizabeth Borowsky and Susie Go and Theodore H. Romer and Ralph Becker-Szendy and Richard Golding and Arif Merchant and Mirjana Spasojevic and Alistair Veitch and John Wilkes. *Minerva: An automated resource provisioning tool for large-scale storage system*. ACM Trans. on Computer Systems, 2001.

[9] J. Wilkes. *Traveling to Rome: QoS specifications for automated storage system management*. Intl. Workshop on Quality of Service (2001) 75-91

[10] Chenyang Lu and John A. Stankovic and Gang Tao and Sang H. Son. *Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms*. Journal of Real-Time Systems, 2002.

[11] C. Lu, G. A. Alvarez, and J. Wilkes. *Aqueduct: online data migration with performance guarantees*. Conference on File and Storage Technologies, 2002.

[12] Christopher Lumb and Arif Merchant and Guillermo Alvarez. *Facade: Virtual Storage Devices with Performance Guarantees*. Conference on File and Storage Technology, 2003.

[13] G.F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems (3rd Ed.)*. Addison-Wesley, 1998.

[14] Gregory R. Ganger, Bruce L. Worthington, Yale N. Patt. *The DiskSim Simulation Environment Version 2.0 Reference Manual*. CMU, 1999.

SAN and Data Transport Technology Evaluation at the NASA Goddard Space Flight Center (GSFC)

Hoot Thompson
Patuxent Technology Partners, LLC
11030 Clara Barton Drive
Fairfax Station, VA 22039-1410
Tel: +1-703-250-3754, Fax: +1-703-250-3742
e-mail: hoot@ptpnow.com

Abstract
Growing data stockpiles and storage consolidation continue to be the trend. So does the need to provide secure yet unconstrained, high bandwidth access to such repositories by geographically distributed users. Conventional data management approaches, both at the local and wide area level, are viewed as potentially inadequate to meet these challenges. This paper explores methods deploying a new breed of Fibre Channel (FC) technology that leverages Internet Protocol (IP) infrastructures as the data transport mechanism, a step towards creating a "storage area network (SAN) grid". These technologies include products using the FC Over IP (FCIP) and the Internet FC Protocol (iFCP) protocols. The effort draws upon earlier work that concentrated on standard FC and internet SCSI (iSCSI) technologies. In summary, the vendor offerings tested performed as expected and provided encouraging performance results. However, their operational readiness still needs to be understood and demonstrated. Installing and configuring the products was reminiscent of the early days of FC with driver and version compatibly issues surfacing once again. Maturity will take some time.

1. Introduction
GSFC, as part of a continuing technology evaluation effort, continues its interest in SAN products and related technologies by evaluating and demonstrating the operational viability of new vendor offerings. Under the auspices of the SAN Pilot, earlier testing has shown the advantages of high-speed transport mechanisms such as FC as well as the flexibility that iSCSI provides in deploying a SAN [1]. Subsequent testing is building upon this work, emphasizing higher speed campus backbones with a focus on manageability as well connectivity to geographically distributed sites. Standardized benchmarks provide measurement of inherent link throughput. In addition, the push is on to attract users with real applications that could benefit from these kinds of technologies

The vision is direct access to data regardless of geographical location, using IP based wide area networks (WAN) as the transport mechanism. Such distributed storage, whether for disaster preparedness or for logical proximity to a compute server, pushes the operational requirements normally associated with direct-attached storage onto the WAN. The storage will be expected to be both reliable and high performance, and to behave like direct attached and physically local. The vision promotes leaving data static and performing the necessary processing directly on a data store as opposed to moving large quantities of data between user facilities. Connections would be temporal in nature with a corresponding service, such as the Storage Resource Broker (SRB) [2], to assist users in

locating relevant data. The end result would be a SAN grid, analogous in many ways to more traditional grids currently gaining wide exposure. This paper explores a variety of topics seen as contributing to the vision.

2. SAN Pilot Infrastructure Description

The core of the SAN Pilot (figure 1) is the connectivity between multiple, on-campus buildings at GSFC. Traditional FC dominates the local GSFC infrastructure with a mix of 2 Gigabit/sec and 1 Gigabit/sec switches – Brocade 3800s and 2400s – providing ports for a variety of server and storage technologies. Linux, Solaris and Apple hosts are represented. RAID storage systems include a DataDirect Networks S2A6000, an Apple Xserve, an Adaptec/Eurologic SANbloc and a Nexsan ATABoy2. A pair of Nishan IPS 3000 Series Multiprotocol IP Storage Switches as well as a LightSand I-8100 augment the other switches by bridging the FC fabric to the IP network. A pair of legacy Cisco SN5420s used for iSCSI work completes the topology. The equipment is mostly GSFC owned. However, notable exceptions include the Nishan and LightSand IP switches. Cisco, Brocade and ADIC have also provided loaner equipment during the testing.

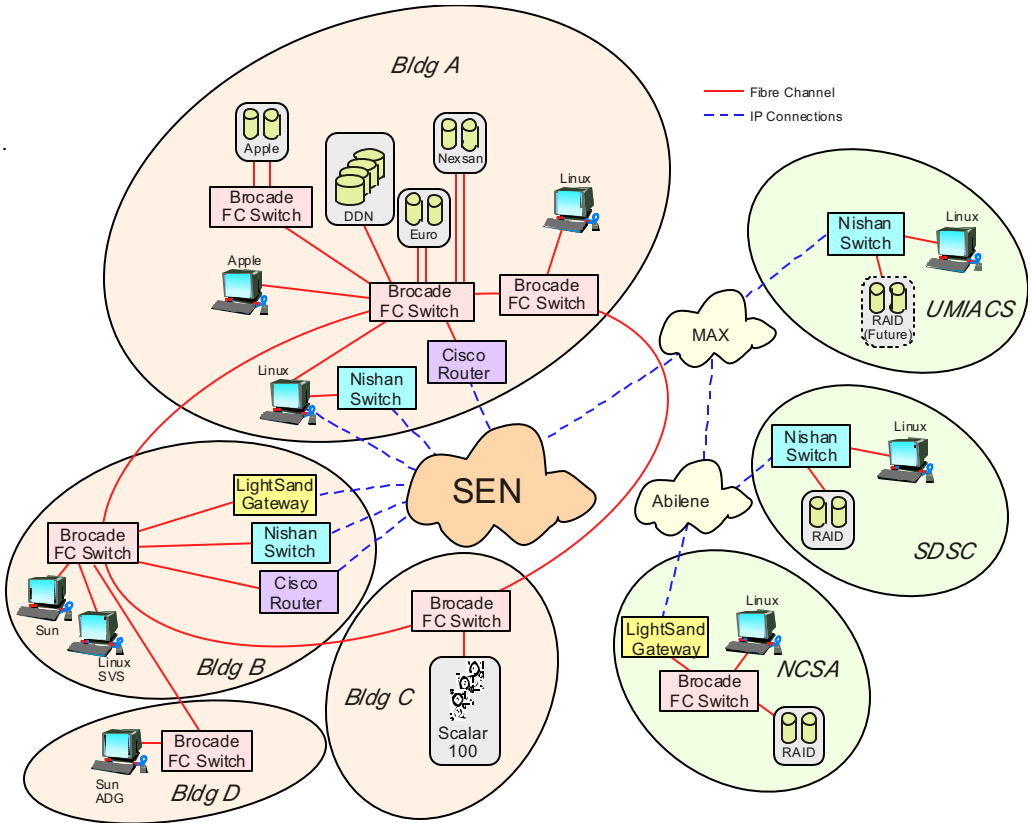


Figure 1 - SAN Pilot Infrastructure

The Nishan and LightSand equipment provide IP connections to similar boxes at the University of Maryland Institute for Advanced Computer Studies (UMIACS), the San Diego Supercomputer Center (SDSC) and the National Center for Supercomputing Applications (NCSA). The underlying networks have been key to the IP related testing. Local to GSFC, the primary backbone is the Science and Engineering Network (SEN)

[3]. Connection to UMIACS is attained by the Mid-Atlantic Crossroads (MAX) [4]. MAX is also the jump off point to the Abilene Network [5] that completes the circuit to both NCSA and SDSC. The result is full Gigabit Ethernet (GE) to all of the remote sites.

2.1. SEN Network

The SEN is a local, non-mission dedicated computer network with high-speed links to the Internet2's Abilene and other Next Generation Internet (NGI) networks. It serves GSFC projects/users who have computer network performance requirements greater than those allocated to the general-use, campus-wide Center Network Environment. The majority of the SEN's inter-building backbone links are 4 gigabits per second (Gbps), created using IEEE 802.3ad link aggregation standards with four separate GE connections between respective pairs of switches. For desktop workstations and servers, as well as for its other inter-building and intra-building links, the SEN minimally provides GE LAN connections. Only jumbo frame-capable GE switches are used in the SEN's infrastructure. The 9000-byte sized Ethernet jumbo frames (maximum transmission unit or MTU) generally provide individual users with approximately six times better throughput performance as compared to networks only supporting standard 1500 MTUs. The SEN presently supports a 2 Gbps jumbo frame-capable link with the MAX point-of-presence at the University of Maryland College Park.

2.2. MAX Network

The MAX is a multi-state metaPoP consortium founded by Georgetown University, George Washington University, the University of Maryland, and Virginia Polytechnic Institute and State University. The proximity of the MAX to Washington, D.C. places it in an advantageous location to partner with federal agencies as well as the business community and post-secondary institutions of DC, Maryland and Virginia. MAX represents a pioneering effort in advanced networking, with the potential to rapidly incorporate a broad cross-section of the not-for-profit community. The MAX, the regional gigapop for access to the Abilene network and the NGI-East Exchange, provides the SEN with excellent WAN connectivity.

2.3. Abilene Network

The Abilene Network is an Internet2 high-performance backbone network that enables the development of advanced Internet applications and the deployment of leading-edge network services to Internet2 universities and research labs across the country. The network supports the development of applications such as virtual laboratories, digital libraries, distance education and tele-immersion, as well as the advanced networking capabilities that are the focus of Internet2. Abilene complements and peers with other high-performance research networks in the U.S. and internationally. The current network is primarily an OC-192c (10 Gbps) backbone employing optical transport technology and advanced high-performance routers.

3. FCIP and iFCP Technology

Prior testing focused on standard FC and iSCSI technologies as it applied to on-campus connections and/or short distances. Interest shifted to assessing the feasibility of constructing a geographically distributed SAN system. This led to experimenting with

more suitable technologies, namely FCIP and iFCP. Several products are available that exploit these protocols. The two tested extensively were the IPS 3000 Series IP Storage Switch by Nishan Systems, now a part of the McData Corporation, and the i-8100 unit by LightSand Communications, Inc. The following paragraphs give a brief overview of each of the products and summarize the current evaluation status.

3.1. Nishan IPS 3000 Series IP Storage Switch

The IPS 3000 and 4000 Series IP Storage Switches use standards-based IP and GE for storage fabric connectivity. Nishan's Multiprotocol Switch supports iSCSI, iFCP, and E_Port for trunking to both IP backbones and legacy FC fabrics. The IPS 3000 Series connects to a wide variety of end systems, including FC, NAS, and iSCSI initiators and targets. The switch has a non-blocking architecture that supports Ethernet Layer 2 switching, IP Layer 3 switching and FC switching over extended distances at full Gigabit wire speed. The Series also supports standard IP routing protocols such as open shortest path first (OSPF) and distance-vector multicast routing protocol (DVMRP) and can be fully integrated into existing IP networks.

Three parameters assist in tuning the performance of the Nishan to a specific environment – Fast Write™ [6], compression [7] and MTU size. When servers and storage are interconnected via a WAN using a pair of Nishans, the normal SCSI exchange (figure 2) required for a 1MB file write will break the data into multiple transfers thereby compounding the "round trip time (rtt)" effect. In contrast, with Fast Write enabled, when the server sends the SCSI write command (figure 3) to set up the transfer, the local Nishan responds with a transfer ready specifying that the entire 1MB of data can be sent at once. At the same time, the sending Nishan forwards the SCSI write command across the WAN so that the target can be prepared to receive data. Having received the 1MB of data from the server, the sending Nishan streams the 1MB block across the WAN to the receiving Nishan. The receiving Nishan, in turn, mimics the normal command/response sequence for the transfers until all of the data is given to the target. The Nishans do not spoof write completion. Instead, the actual status generated by the storage target is passed back through the network to the server. This guarantees that all data was actually written to disk.

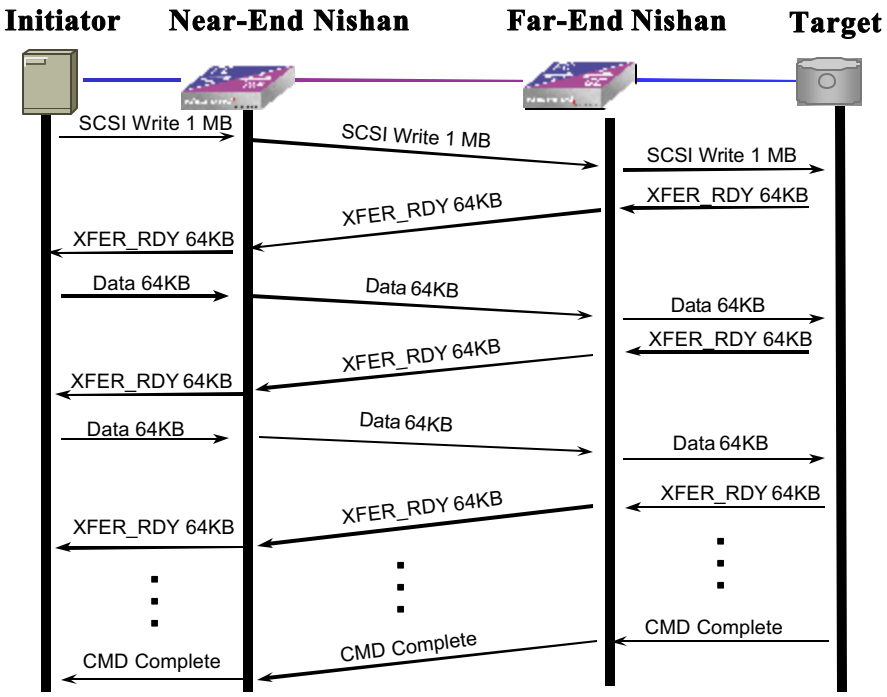


Figure2 - Normal SCSI Exchange for a 1MB Write

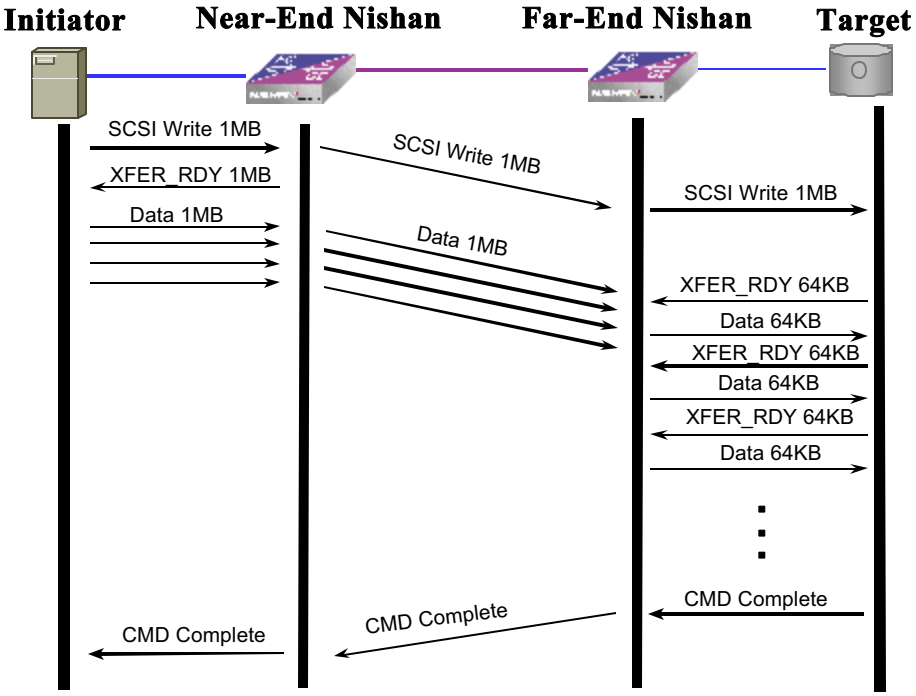


Figure 3- Fast Write Modified SCSI Exchange

The Nishan switch also features software based *lossless* compression. The following options are available:

- **Off** - Data going out of the port is not compressed.
- **On** - Data going out of the port is always compressed using the appropriate algorithm to achieve maximum compression.
- **Auto** - Depending on the available bandwidth, the switch dynamically decides whether or not to compress the data, the level of compression to apply and the compression algorithm to use. With the Auto setting, the port keeps the data rate as close as possible to the Port Speed of the port.

The last key parameter is MTU. The Nishan switches can support packet sizes up to 4096 bytes, an increase of almost 3X over the nominal 1500. The larger data payload results in less header processing overhead and better link utilization. Packet sizes greater than 1500 bytes maximizes direct matching with FC originated frames. The full FC data payload of 2112 bytes can be delivered in a single jumbo, 4096 byte Ethernet frame. The "auto" option for MTU setting allows Nishan switches to negotiate the best possible rate.

Configuring the Nishan switch involves the interaction of two applications, the switch resident http GUI Element Manager and the host based (Linux or Solaris) SANvergence Manager application. Between the two, devices to be shared are placed in commonly seen, exported zones. The level of SAN merging is a cooperative effort between two or more switches. As a default, a CLI is also available.

3.2. LightSand i-8100

The LightSand i-8100A is an intelligent gateway that provides connectivity between FC fabrics across an IP WAN infrastructure. The i-8100A is an eight port, multi-protocol switch that provides isolation between FC SANs using Autonomous Region (AR) technology. Conventional FCIP bridging devices link two sites by merging the FC fabrics together. By maintaining Autonomous Regions, the i-8100A is able to share storage devices without merging fabrics. In the diagram (figure 4), two autonomous regions are joined. Each AR consists of four FC switches, the three original switches plus the gateway. If these two SANs had been bridged by a simple FCIP gateway (non-switching), the fabric would appear as six FC switches—all part of the same fabric. The storage arrays labeled Disk 1 and Disk 2 are shared. Once they have been imported into SAN 2, every initiator in SAN 2 can see the shared disks as if they were present in SAN 2. In reality, the I-8100A is performing Domain Address Translation (DAT) and the actual disks remain inside SAN 1. Because of this technology, each fabric is isolated from any disturbances that might occur in the other fabric.

The LightSand i-8100A employs the user datagram protocol (UDP) with an additional sequencing number to enable protection against packet-loss and mis-ordering. This protocol is referred to as UDP/SR (UDP with Selective Retransmission). Using UDP/SR, the i-8100A can be set for a desired WAN bandwidth. It will instantly jump to that bandwidth and execute appropriate backpressure against the FC fabric, if the WAN bandwidth is less than the native FC bandwidth. In the event that there is packet-loss on the WAN, the i-8100A will retransmit the lost data without throttling the bandwidth.

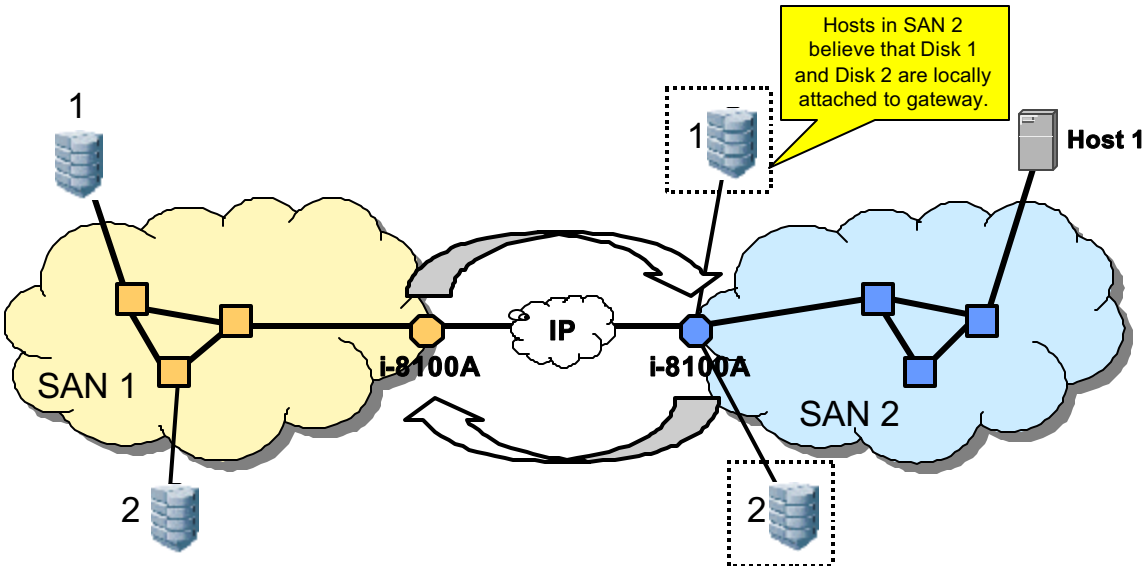


Figure 4- LightSand Interconnect

Configuring the LightSand switch requires running the SANman GUI on each of switches or using the available CLI.

3.3. EvaluationProcess andResults

As evidenced by the work done at SDSC for last year's Mass Storage conference [8], outstanding performance moving data over IP is achievable using a well-behaved, highly tuned network. The tact taken at GSFC has been more the "every day", out-of-the-box approach where nothing aggressive is done to enhance the performance of site-to-site WANS. In more real world networks, the effects of rtt, congestion and packet loss can render an application useless that requires high bandwidth. In the spirit of the SAN grid vision, laying a distributed file system, such as ADIC's StorNext File System (SNFS) or SGI's CXFS™, on the topology would further attenuate any irregularities.

FCIP and iFCP testing has been a multi-step process:

- Evaluate the technology on a local, campus basis under ideal network conditions.
- Artificially introduce non-zero rtts, packet loss and congestion into the circuit, and observe the impact on performance.
- Connect to a geographically distant center(s) and compare performance to predictions based on simulated distance testing.

Testing was performance centered using standard benchmarks such *lmdd* [9] and *IOzone* [10] as the primary tools. *lmdd* is good for quick, single threaded operations. *IOzone* permits a variety of IO operations including writes, reads, mixed writes and reads, multi-threaded operations, etc. all with options for setting attributes such as record and file size. The majority of the tests consisted of multiple *IOzone* operations described by the following script:

./iozone_mod -i 0 -i 1 [-+d] -r 1m -s 16g -b one_thread

./iozone_mod -t 2 -i 0 -i 1 [-+d] -r 1m -s 8g -b two_threads
./iozone_mod -t 4 -i 0 -i 1 [-+d] -r 1m -s 4g -b four_threads
./iozone_mod -t 8 -i 0 -i 1 [-+d] -r 1m -s 2g -b eight_threads

The scripts steps through 1, 2, 4 and 8 threaded write/read operations and in aggregate moves 16 Gbytes. *IOzone* was modified such that the [-+d] option would generate random data without doing the diagnostic byte-for-byte check of the data. This was done to evaluate the efficiency of the Nishan compression algorithm while not impacting performance with verification process. Tests were performed using mostly native file systems (ext2) with some minimal SNFS evaluation.

Network utilization was also monitored. Data traffic cannot be at the expense and disruption of existing communication traffic. At a minimum, the impact must be understood and anticipated. Nishan and LightSand use two different approaches to how the data is transported so the resulting network perturbation varies.

3.3.1. On-Campus Testing

Testing began at GSFC with a pair of Nishan switches. A Linux machine was FC connected to one of the Nishans co-located in the same building (figure 5). The other Nishan, in a different building provided tie-in to the SAN Pilot and its associated RAID. Initial results, with zero rtt, compared favorably with the same tests using directly connected RAID.

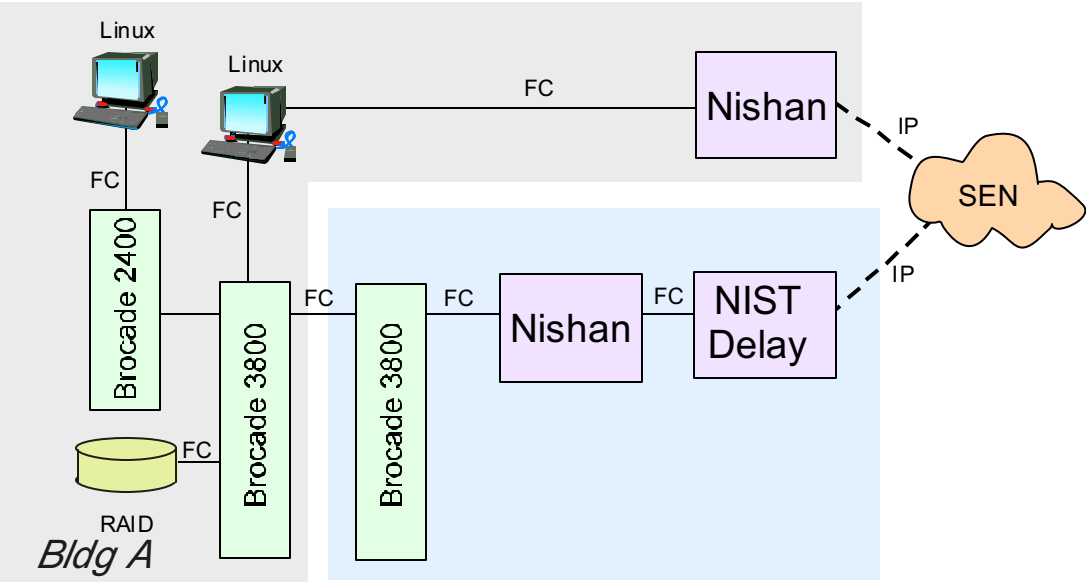


Figure 5- Local GSFC Testing

The next step was to introduce set delays into the circuit using a NIST Net [11] network emulator to simulate the potential effects of geographically separating the two Nishan switches. The NIST Net network emulator is a general-purpose tool for emulating performance dynamics in IP networks. The tool is designed to allow controlled, reproducible experiments. By operating at the IP level, NIST Net can emulate the critical end-to-end performance characteristics imposed by various WAN situations (e.g.,

congestion loss) or by various underlying subnetwork technologies (e.g., asymmetric bandwidth situations of xDSL and cable modems).

Impressions

Installation and configuration of the Nishan units was relatively straightforward with the assistance of the product support engineers. Besides providing FC-IP translation, the Nishans are also full FC switches, an attribute that has different ramifications depending upon how the device is introduced into an existing SAN. As a standalone switch with directly connected devices, as was the case for one end of the GSFC circuit, operation was clear with only the usual zoning decisions to be made. The second switch was E-port connected, a more complicated configuration which requires choosing how the Nishan was to interoperate with the existing SAN Pilot Brocade infrastructure. Multiple options are available, so the ripple effect of zone changes, for example, need to be understood to avoid any unforeseen interruption of an operational SAN. Setting up the zones and mapping devices was easily accomplished using SANvergence and the Element Manager.

Large transfers (files) were required to overcome the buffering effects of the servers, the switches and the link. With *IOzone* modified accordingly, a variety of tests were executed varying rtt and MTU size while going through the permutations of the Fast Write and compression settings. Three observations were made:

- Fast Write seems to have an overall positive effect on write performance with this likely being the default setting. Nishan recommends setting to "on" for distances over 200km noting potential degradation if "on" for shorter distances.
- Compression can have a positive or negative effect depending upon rtt. Compression processing significantly reduces throughput when rtt is small. Conversely, for large rtt compression enhances performance. Nishan recommends the "auto" mode letting the switch dynamically determine the appropriate level of compression.
- The effect of increasing MTU size from 1500 to 4096 was somewhat inconclusive but an odd jump was noted when both FastWrite and compression were turned "off". Intuitively the larger frames should improve performance but the suspicion is that the effects of a large rtt on the SCSI exchange may mitigate the gain. This warrants further testing.

In summary, settings are situation dependent. This warrants exercising all the combinations before finalizing an installation. To illustrate the point, the following graphs (figure 6 and 7) depict bandwidth as a function of threads for rtt=35msec for different MTUs, Fast Write and compression settings. For MTU = 1500, the best write performance was for Fast Write, no compression while read was best for Fast Write with compression enabled. Bumping the MTU to 4096 resulted in both the write and read numbers being best with Fast Write and compression disabled. Incidentally, these parameters are changed using the Element Manager with each switch configured independently. The implication is that unpredictable results may occur if the switches are not configured the same. Overall, the write performance topped out at just slightly over

25 MB/sec while read approached 20MB/sec. For the most part, running multiple threads

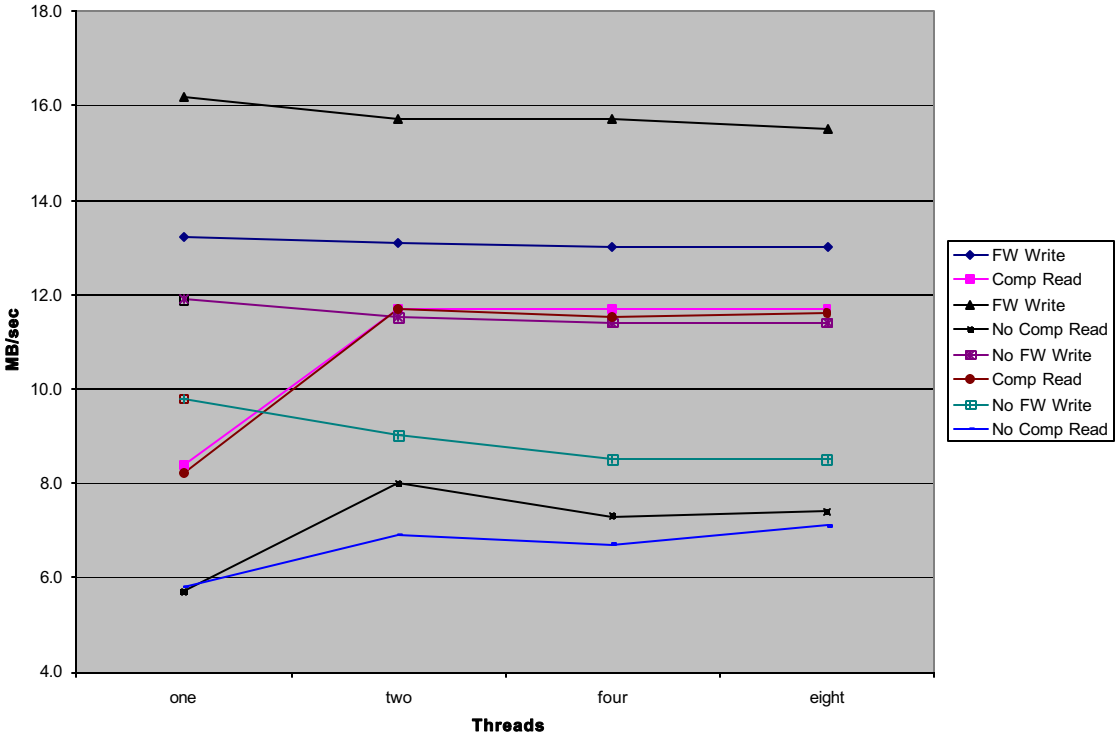


Figure 6 - Delay=35msec, MTU=1500

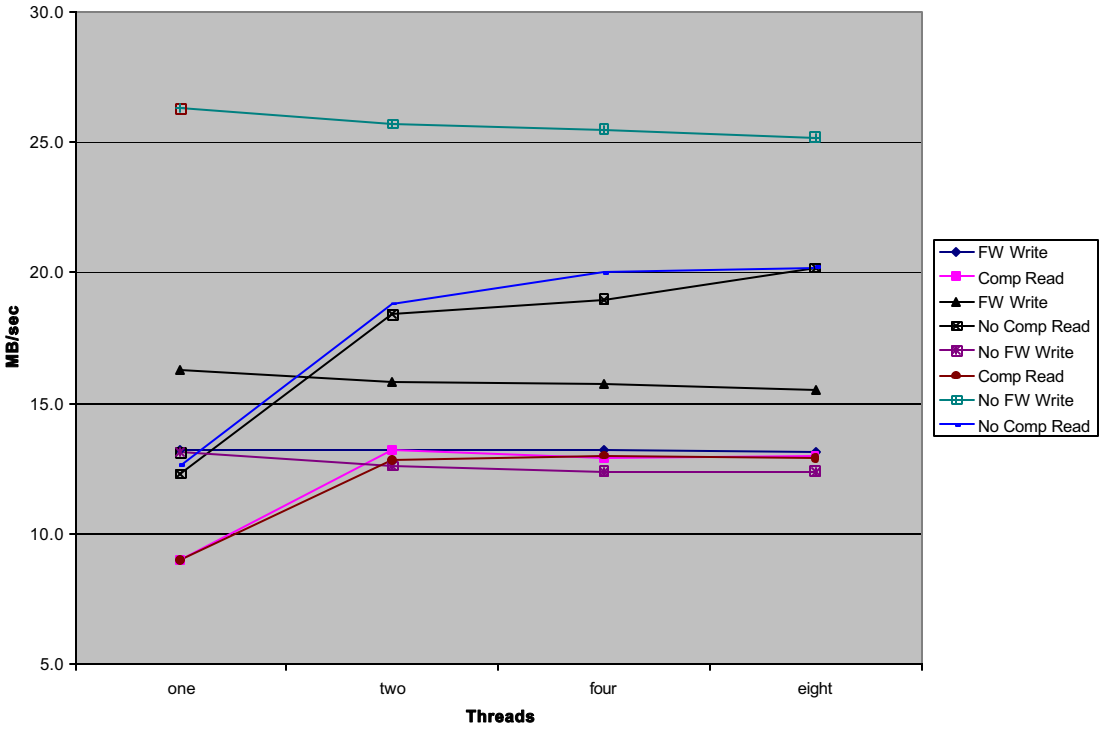


Figure 7 - Delay=35msec, MTU=4096

boosted aggregate throughput. These numbers are in contrast to 86 MB/sec writes and 78 MB/sec reads obtained running eight threads with rtt=0, MTU=1500 and both Fast Write and compression turned off.

Future Testing

Additional tests to be conducted include:

- Run tests with a broader range of rtt values while changing configuration of the Nishan units. This would give the full curve for bandwidth as a function of rtt.
- Test the compression "auto" setting in contrast to the "on/off" results.
- Induce deterministic packet loss and congestion, and measure the impact on write and read performance.

3.3.2. Multi-site Testing

The next series of tests involved different combinations of IP hardware and network connections to UMIACS, SDSC and NCSA. Experiments focused mainly on building and exercising native file systems (ext2) with server/host and storage at opposite ends of the WAN link. Some preliminary SNFS testing was also accomplished. In all cases, the assessment centered on:

- Gauging the impact of rtt or latency on performance in a real world setting where the network is potentially hostile.
- Comparing measured maximum network bandwidth, as determined using nuttcp, with file system oriented traffic.

3.3.2.1. UMIACS

Last year, UMIACS participated with GSFC in distance testing using iSCSI technology. That effort involved a Linux box at UMIACS routed through a Cisco SN5420 at GSFC to the associated storage assets. This time for comparison, one of the two loaner Nishan units was moved to UMIACS (figure 8). Nishan-to-Nishan communication was established using the MAX network. *IOzone* benchmarks were performed building a native ext2 file system on GSFC storage from an UMIACS resident Linux host.

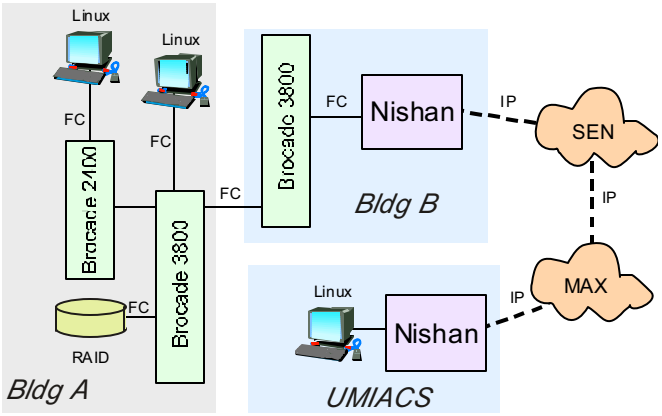


Figure8 - GSFC - UMIACS Configuration

Impressions

Moving and establishing the Nishan to UMIACS connection was relatively simple. Network logistics provided the only significant obstacles. Getting the Nishan configuration tools functioning in a new environment posed a minor nuisance. Only certain browser/host combinations will run the Element Manager GUI. Secondly, UMD, except in specific instances, blocks SNMP which led to establishing a virtual private network (VPN) for remote access to both Element Manager and SANvergence.

Performing *IOzone* testing with random data yielded the following results (Table 1) for one, two, four and eight threaded operations. These results are for an MTU size of 1500 and a negligible rtt as registered by the Nishans.

Table 1 - Results

	FW, Comp		No FW, No Comp	
Threads	Write	Read	Write	Read
one	12.8	9.5	38.6	14.1
two	12.9	11.7	47.3	19.8
four	12.8	11.6	28.9	20.6
eight	12.8	11.6	59.8	25.8

Given the near zero rtt, the boxes ran best with both Fast Write and compression disabled. As noticed in other testing involving the Nishan, compression processing effectively halves the bandwidth in applications involving small rtts. The eight threaded write, 59.8 MB/sec, saturated the network given the available bandwidth, as measured by nuttcp [12], was 56.2MB/sec. Reads topped out at 25.8MB/sec. Single threaded *IOzone* tests saw 38.6MB/sec writes and 14.1MB/sec reads. As it turns out, the WAN connection at UMIACS end is not full GE but rather a fractional allocation of a full GE. By comparison to historical data, single threaded iSCSI operations using lmdd yielded 18MB writes and 12MB reads.

Future Testing

Additional tests to be conducted include:

- Increase network bandwidth between GSFC and UMIACS to a full GE and reevaluate Nishan performance. Given the almost negligible rtt, a significant performance jump is anticipated.
- Connect storage to the UMIACS Nishan then test reads and writes originating at GSFC.
- Exercise the UMIACS-to-SDSC connection and compare to the GSFC-to-SDSC results.

3.3.2.2. SDSC

Testing with SDSC (figure 9) leveraged the in-place, SDSC Series 4000 switch. WAN

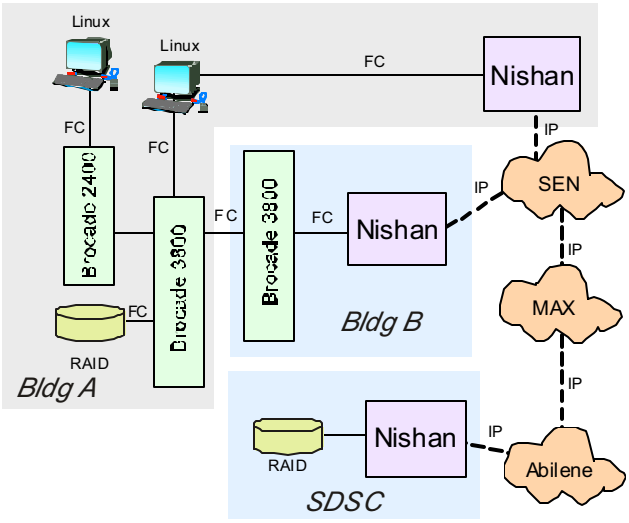


Figure 9– SDSC Configuration

connection used the Abilene backbone with MAX as the local hopping off point for GSFC. *IOzone* benchmarks were performed building a native ext2 file system on SDSC Sun storage from a GSFC resident Linux host.

Impressions

Set-up was straightforward with only the expected configuration items to be dealt with, namely network routing and allocating the appropriate zones, resolving SAN IDs, etc. However, the switches could not be made to operate in the jumbo frame (MTU=4096) mode, although the network was theoretically configured for such operation. It was learned though trial and error that manually forcing the MTU setting to 4096 can result in very erratic behavior of the link including complete lock up. The next two graphs (figures 10 and 11) illustrate performance as a function of the various Nishan settings for random versus static data.

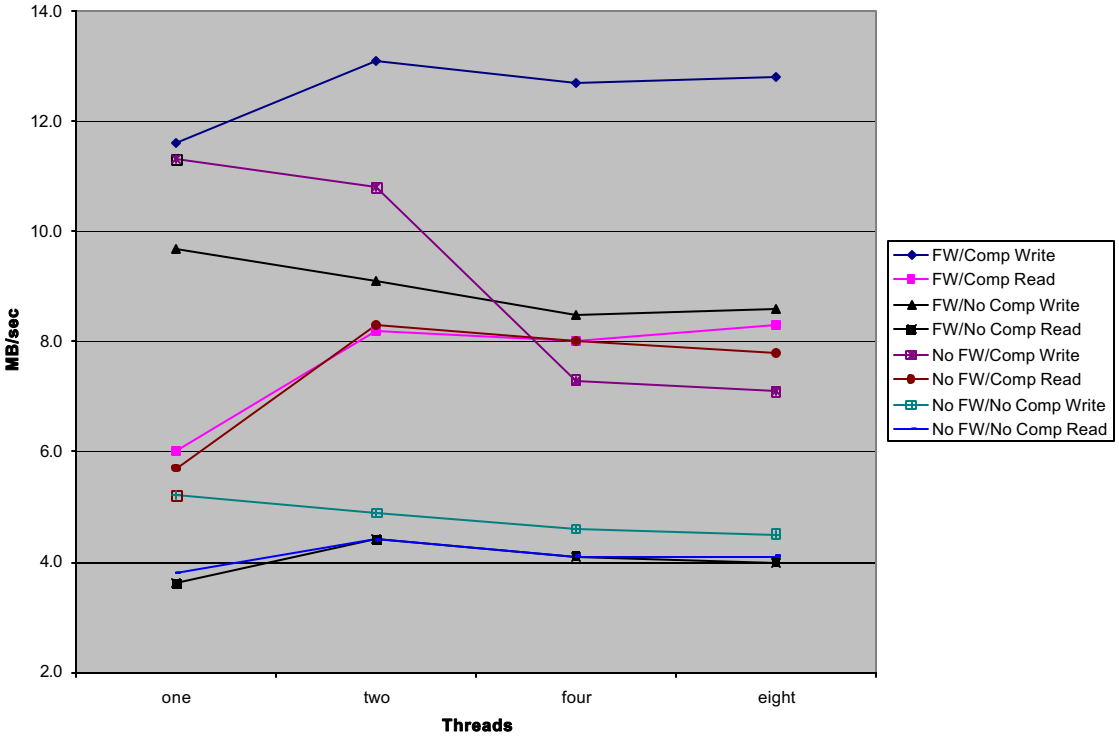


Figure10 - Random Data, MTU=1500

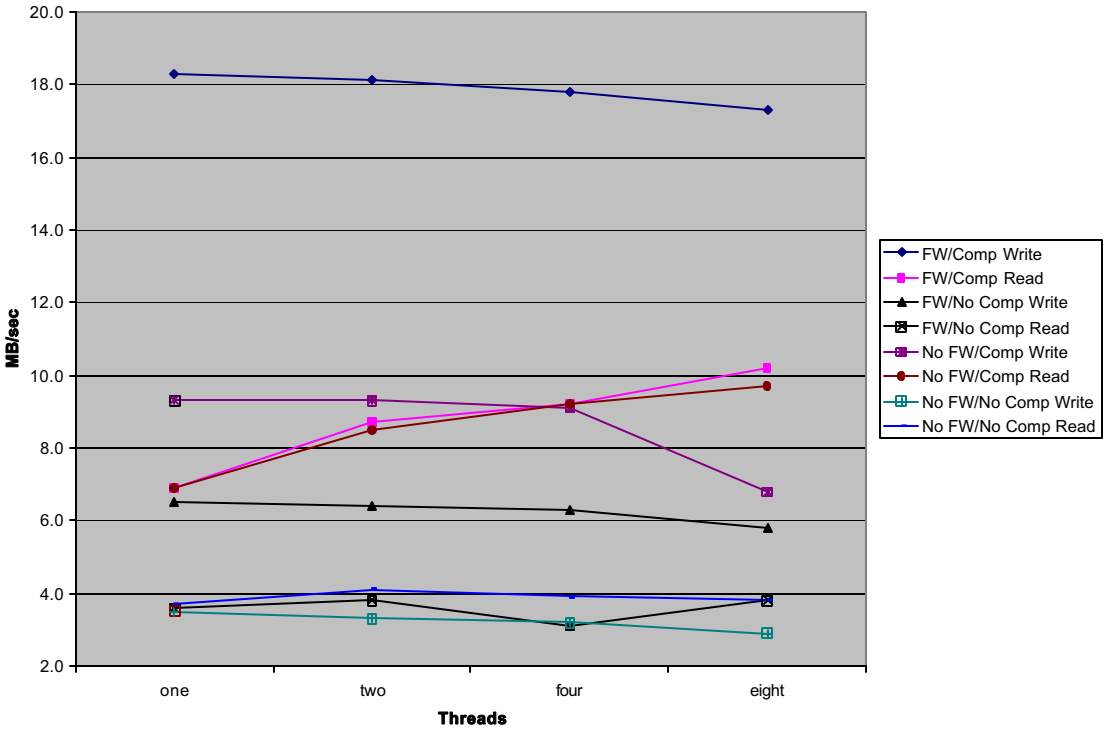


Figure11 - Static Data, MTU=1500

The following data (Table 2) compares actual results of the GSFC-to-SDSC connection with test data using the NIST simulator with an equivalent rtt of 70msec. In both cases, Fast Write and compression are turned on. Note fair agreement in the data despite the difference in MTU sizes. The suspicion is that the rtt impact on the SCSI command interchange dilutes the performance gains of jumbo frames.

Table 2 - Results

	GSFC => GSFC rtt delay => 70msec MTU => 4096		GSFC => SDSC rtt actual => 70msec MTU => 1500	
Threads	Write	Read	Write	Read
one	13.1	5.6	11.6	6.0
two	13.1	11.5	13.1	8.2
four	13.1	12.5	12.7	8.0

Future Testing

Additional tests to be conducted include:

- Get jumbo frames (MTU=4096) working between GSFC and SDSC then reevaluate performance and compare to delay numbers. Determine if the jump in performance was an anomaly related to the NIST emulator.
- Exercise link in opposite direction – server/host at SDSC and storage at GSFC.
- Exercise the SDSC-to-UMIACS connection and compare to SDSC-to-GSFC results.

3.3.2.3. NCSA

The IP connection with NCSA (figure 12) was accomplished using a pair of LightSand i-8100s. As with SDSC, WAN connection used the Abilene backbone with MAX as the local hopping off point for GSFC. *IOzone* benchmarks were performed building a native ext2 file system on NCSA DataDirect storage from a GSFC resident Linux host.

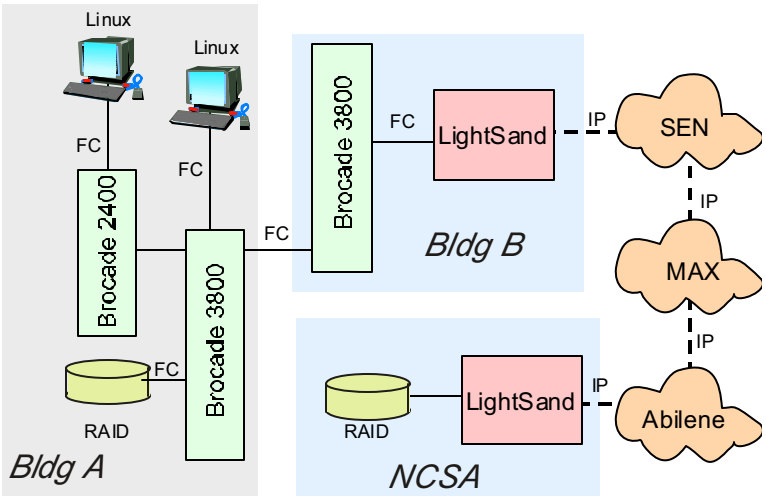


Figure 12 - NCSA Configuration

Impressions

Initial set-up was time consuming because of the learning curve of dealing with the LightSand equipment and establishing the network connection between GSFC and NCSA. The LightSands required that the Brocade 3800 switches be at the 3.1 firmware level. In addition, the command "portcfgislmode <port>,1" also had to be issued to the Brocades so that the switch ports connected to the 8100s would get the R_RDY set. An inordinate amount time was spent trying to determine why the SANman GUI would not execute properly from a remote workstation (off campus with respect to GSFC). As it turns out, NASA blocks external pings from open networks and the first thing the LightSand GUI requires is a successful ping to make sure the connection is in place. Once properly configured, the DataDirect Networks storage at NCSA was easily configured and accessed. Using the same *IOzone* script as before, the following results (Table 3) where obtained for native, ext2 file transfers.

Table 3 - Results

rtt => 30msec 1MB block

Linux Host 1		
Threads	Write	Read
one	37.0	12.1
two	37.5	28.9
four	37.3	35.6
eight	37.3	36.2

These numbers are consistent with the theoretical maximums as predicted by the TimeCalc utility provided with the SANman. An interesting although not perfect comparison is the 35msec rrt numbers obtained using the NIST Net network emulator and Nishan switches. The best results with Fast Write and Compression turned off, was 26MB/sec writes and 20 MB/sec reads. It seems fair to presume, that running the Nishans in the "auto" compression mode may have improved those results.

Future Testing

Additional tests to be conducted include:

- Exercise link in opposite direction – server/host at NCSA and storage at GSFC.
- Get raw bandwidth numbers for the GSFC to NCSA link using nuttcp.

4. Operational Users

As to what might seem like a sidebar to the major thrust of the evaluation, the search for a relevant application of this technology, a geographically distributed file system, continues. Two GSFC groups, the Scientific Visualization Studio (SVS) and the Advanced Data Grid (ADG) Project, are currently being pursued to provide on-campus operational proof of the various connectivity schemes. The plan is to also involve UMIACS, SDSC and NCSA in relevant application demonstrations.

4.1. Scientific Visualization Studio

The GSFC SVS has a need for approximately 1 TB of storage to use as an animation "scratch" area. The content/data to be stored will be scientific visualization animation frames in both HDTV and NTSC resolutions, and MPEG-1 and MPEG-2 movies in various resolutions from web to HDTV. Relatively fast (high bandwidth) access to such volumes is required, including constantly writing frames, various types of processing (read/write) of frames, and streaming frames from this volume to the local SVS workstations for animation preview. A Linux server in the SVS has an FC connection to the SAN Pilot.

4.2. Advanced Data GRID Prototype

In conjunction with NASA Ames, the ADG prototype is a new initiative that intends to leverage the availability of Landsat data The mechanism for making the data available is the SAN Pilot connected to a Sun 3800 located on the GSFC campus.

5. Supporting Technologies

Other technologies are being evaluated to ease the administrative burden of SANs as well improve the performance of the chosen data transport mechanism. The list includes SAN management software and a new generation of network interface (NIC) cards. Also, the evolution of network attached storage (NAS) is also being monitored.

5.1. SAN Management Software

With the emphasis on connecting operational users, part of the testing has focused on SAN management software and tools. The goal is to acquire a tool or suite of tools that enables efficient monitoring of the SAN health and utilization as well as providing for asset allocation and administration. A mechanism is needed that readily discovers SAN components and provides a topology view of the infrastructure.

Four such tools have been installed and evaluated:

- BrightStor™ SAN Manager by Computer Associates International, Inc.
- SANavigator® by SANavigator, Inc. a subsidiary of McData Corporation
- SANScreen by Onaro, Inc.
- Fabric Manager and WEB TOOLS by Brocade Communications Systems, Inc.

The shortcoming of all such products seems to be coverage of all the needed versions of operating systems, and storage and interface devices, something not usually supported. Recognizing the new breed of FC and FC related products, such as Nishan and LightSand boxes, is sporadic as well. No one product seems to do it all. Not tested but briefed was a StorageAuthority™ Suite from AppIQ, Inc. It possesses some very rich capabilities worthy of consideration. In the meantime, SANScreen was purchased and installed. It will be important to observe how the product deals with a heterogeneous, near operational environment with ever evolving security constraints.

5.2. NIC Evaluation

This testing is most relevant to iSCSI connected hosts. The plan is for parametric evaluation of generic NICs versus TCP Off-Load Engine (TOE) NICs and TOE iSCSI NICs. It will be key to measure end-to-end throughput performance and CPU utilization on hosts with different processor speeds. The intent is to include cards from multiple manufacturers such as Intel, Adaptec, and Alacritech. Testing is underway but not yet completed. So far, getting the basic set-up configured and operational is proving to be a challenge.

6. Summary

In retrospect, the testing permutations became formidable when the multiple locations, potential rtt, equipment configurations and settings are factored in. As a result, only a subset of possible hardware and software combinations were actually exercised. However, the size of the data sampling does not adversely impact the overall evaluation of the products. Evaluating IP devices has been an educational process punctuated by learning new jargon and redefining the concept of a SAN while dealing with the unavoidable reality of the hardware and software incompatibilities, typical of emerging technology. This class of product is mainly deployed in disaster recovery applications as opposed to file system applications. As a result, empirical data for comparison was not readily available, leaving conversations and paper exercises as the basis for determining the validity of the collected data. A better understanding of theoretical maximums as they relate to SCSI transfers as a function of rtt versus the selected FC-IP protocol (FCIP or iFCP) is needed.

The vendor products behaved admirably with one significant, non-performance concern. Security features were found to be lacking from a device management perspective – no secure login, clear text passwords, etc. To circumvent such shortfalls during the testing, network routing was altered and access lists were incorporated to minimize the perceived vulnerabilities. Also, a desirable feature available at the data level for iSCSI is host authentication by the IP interface. The following table (Table 4) presents a qualitative review of the Nishan and LightSand equipment:

Table 4 – Findings Summary

IP Device	Pros	Cons
General	• Perform as advertised. • Operationally fairly intuitive. • Both GUI and CLI management options. • Administrator defined level of SAN merging/isolation.	• Minimal security. • No ssh. • No CLI standard • Redundant, conflicting naming conventions. • Proprietary, same vendor product required at both ends of the WAN connection. • High skill level to configure, etc., multiple talents involved. • Incompatibilities, version issues, etc. reminiscent of the early days of FC.
Nishan 3000	• Built in performance graphs. • Good statistical info.	• Passwords in clear text.
LightSand i-8100	• Companion applications that provide data analysis.	• IP routes cleared by reboots. • Difficult to save and compare configurations.

A sidebar to the qualitative aspects of the testing is that the majority of configuration, benchmarking, etc. was done remotely from third party locations, not at any of the centers. Besides the obvious advantage of permitting geographic flexibility for the testers and vendors, it had the interesting side effect of revealing obstacles to deploying such a methodology for an operational IP based SAN. In place site security procedures and firewalls had to be acknowledged and understood. Blocked ports and disabled functionality had to be navigated. Such activity led to a greater understanding of the equipment and what changes would be welcomed in the products.

Certainly at one level the objective of the testing was met – to gain experience with data over IP devices. Understanding the requirements being levied against a proposed SAN has always been critical, but the extra layer of configuration encountered installing FC-IP devices makes such planning even more necessary. There is the usual FC zoning at the local SAN level but in addition, bridging disparate SANs requires designating which components – servers, storage, etc. – will be mutually shared by the co-joined SANs. This two-step mechanism, while adding to the rigor, ensures isolation and privacy of the local SAN while allowing the sharing of mutually agreed to assets. Plans fell short in terms of evaluating a geographically distributed file system (SNFS and/or CXFS) encompassing GSFC, UMIACS, SDSC and NCSA, an outcome planned to be rectified in the near future. These file systems have centralized agents that control their overall operation. It will be interesting to track data movement performance (throughput) as a function of where in the topology the agent is located and the latencies incurred in accessing it.

Acknowledgements
The author wishes to acknowledge the following individuals for their contributions: Bill Fink, Paul Lang, Wei-Li Liu and Aruna Muppalla at NASA GSFC; Bryan Bannister and Nathaniel Mendoza at SDSC; Chad Kerner at NCSA; and Fritz McCall at UMIACS. Gratitude is also extended to the vendor community for their rich support.

References

[1] Hoot Thompson, Curt Tilmes, Robert Cavey, Bill Fink, Paul Lang, Ben Kobler; Architectural Considerations and Performance Evaluations Of Shared Storage Area Networks at NASA Goddard Space Flight Center; Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies; April 7-10, 2003.
[2] http://www.npaci.edu/DICE/SRB/
[3] J. P. Gary; Research and Development of High End Computer Networks at GSFC, Earth Science Technology Conference, College Park, MD; June 24-26, 2003.
[4] http://www.maxgigapop.net/
[5] http://abilene.internet2.edu/
[6] Maximizing Utilization of WAN Links with Nishan Fast Write; Nishan Systems.
[7] FAQ on Nishan Systems' Compression Technology; Nishan Systems.
[8] Phil Andrews, Tom Sherwin, Bryan Bannister; A Centralized Data Access Model for Grid Computing; Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems & Technologies; April 7-10, 2003.
[9] http://www.bitmover.com/lmbench/
[10] http://www.iozone.org
[11] http://snad.ncsl.nist.gov/itg/nistnet/
[12] ftp://ftp.lcp.nrl.navy.mil/pub/nuttcp/beta/nuttcp-v5.1.1.c

File System Workload Analysis For Large Scale Scientific Computing Applications

Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long
Storage Systems Research Center
University of California, Santa Cruz
Santa Cruz, CA 95064
{cyclonew, qxin, hongbo, sbrandt, elm, darrell}@cs.ucsc.edu
Tel +1 831-459-4458
Fax +1 831-459-4829

Tyce T. McLarty
Development Environment Group/Integrated Computing and Communications
Lawrence Livermore National Laboratory
Livermore, CA 94551
{tmclarty@llnl.gov}
Tel +1 925-424-6975
Fax +1 925-423-8719

Abstract

Parallel scientific applications require high-performance I/O support from underlying file systems. A comprehensive understanding of the expected workload is therefore essential for the design of high-performance parallel file systems. We re-examine the workload characteristics in parallel computing environments in the light of recent technology advances and new applications. We analyze application traces from a cluster with hundreds of nodes. On average, each application has only one or two typical request sizes. Large requests from several hundred kilobytes to several megabytes are very common. Although in some applications small requests account for more than 90% of all requests, almost all of the I/O data are transferred by large requests. All of these applications show bursty access patterns. More than 65% of write requests have inter-arrival times within one millisecond in most applications. By running the same benchmark on different file models, we also find that the write throughput of using an individual output file for each node exceeds that of using a shared file for all nodes by a factor of 5. This indicates that current file systems are not well optimized for file sharing.

1. Introduction

Parallel scientific applications impose great challenges on not only the computational speeds but also the data-transfer bandwidths and capacities of I/O subsystems. The U.S. Department of Energy Ac-

celerated Strategic Computing Initiative (ASCI) projects computers with 100 TeraFLOPS, I/O rates of 50–200 gigabytes/second, and storage system capacities of 0.5–20 PB in 2005. The projected computing and storage requirements are estimated to 400 TeraFLOPS, 80–500 gigabytes/second, and 3–20 PB in 2008 [2]. The observed widening disparity in the performance of I/O devices, processors, and communication links results in a growing imbalance between computational performance and the I/O subsystem performance. To reduce or even eliminate this growing I/O performance bottleneck, the design of high-performance parallel file systems needs to be improved to meet the I/O requirements of parallel scientific applications.

The success of file system designs comes from a comprehensive understanding of I/O workloads generated by targeted applications. In the early and middle 1990s, significant research efforts were focused on characterizing parallel I/O workload patterns and providing insights on parallel system designs [1, 4, 7, 14]. The following decade has witnessed significant improvements in computer hardware, including processors, memory, communication links, and I/O devices. At the same time, systems are scaling up to match the increasing demands of computing capability and storage capacity. This advance in technologies also enables new scientific applications. Together these changes motivate us to re-examine the characteristics of parallel I/O workloads a decade later.

In our research, we traced the system I/O activities under three typical parallel scientific applications: the benchmark *ior2* [6], a physics simulation, *f1*, running on 343 nodes, and another physics simulation, *m1*, running on 1620 nodes. We study both static file system and dynamic I/O workload characteristics. We use the results to address the following questions:

- What were the file sizes? How old were they?
- How many files were opened, read, and written? What were their sizes?
- How frequent were typical file system operations?
- How often did nodes send I/O requests? What were the request sizes?
- What forms of locality were there? How might caching be useful?
- Did nodes share data often? What were the file sharing patterns?
- How well did nodes utilize the I/O bandwidth?

The remainder of this paper is organized as follows: a brief overview of the related work is given in Section 2. We then describe the tracing methodology in Section 3 and present our results in Section 4. Finally, we conclude our paper in Section 5.

2. Related Work

The I/O subsystem has been a system performance bottleneck for a long time. In parallel scientific computing environments, the high I/O demands make the I/O bottleneck problem even more severe. Kotz and Jain [3] surveyed impacts of I/O bottlenecks in major areas of parallel and distributed systems and pointed out that I/O subsystem performance should be considered at all levels of system design.

Previous research showed that the I/O behavior of scientific applications is regular and predictable [7, 9]. Users have also made attempts to adjust access patterns to improve performance of parallel file systems [13].

There are several studies on file system workload characterizations in scientific environments [1, 4, 7, 8, 11]. They have shown that file access patterns share common properties such as large file sizes, sequential accesses, bursty program accesses, and strong file sharing among processes within a job. A more recent study [14] showed that applications use a combination of both sequential and interleaved access patterns and all I/O requests are channeled through a single node when applications require concurrent accesses; we observe similar phenomena in one of the applications under our examinations.

Pasquale and Polyzos [9] found that the data transfer rates ranges from 4.66 to 131 megabytes/sec in fifty long-running large-scale scientific applications. They also demonstrated that the the I/O request burstiness is periodic and regular [10].

Baylor and Wu [1] showed that the I/O request rate is on the order of hundreds of requests per second; this is similar to our results. They also found that a large majority of requests are on the order of kilobytes and a few requests are on the order of megabytes; our results differ in this regard.

Previous research has mainly investigated scientific workloads in the 1990's, although technology has evolved very quickly since then. We observed changes in large-scale scientific workloads, in our study, and provided guidelines for future file system designs based on a thorough understanding of current requirements of large-scale scientific computing.

3. Tracing Methodology

All the trace data in this study was collected from a large Linux cluster with more than 800 dual processor nodes at the Lawrence Livermore National Laboratory (LLNL). A development version of Lustre Lite [12] is employed as the parallel file system and the Linux kernel in use is a variant of 2.4.18.

3.1. Data Collection

Tracing I/O activities in large scale distributed file systems is challenging. One of the most critical issues is minimizing the disturbance of tracing on the system behaviors. A commonly-used method is to develop a trace module that intercepts specific I/O system calls—a dedicated node in the cluster collects all trace data and stores them to local disks.

However, due to time limits, we chose a simpler approach: we employed the *strace* utility with parameters tuned for tracing file-related system calls. The trace data are written to local files. We rely on the local host file systems to buffer trace data.

This approach has two shortcomings: first, strace intercepts all I/O-related activities, including parallel file system, local file system, and standard input/output activities. This results in relatively large data footprint. Second, the strace utility relies on the local file system to buffer traced data. This buffer scheme works poorly when the host file system is under heavy I/O workloads. In such a scenario, the host system performance might be affected by the frequent I/Os of the traced data.

However, the strace utility greatly simplifies the tedious data collection process to a simple shell script. More importantly, the shortcomings mentioned above were not significant in our trace col-

Table 1. The ASCI Linux Cluster Parameters

Total Nodes (IBM x355)	960
Compute Nodes	924
Login Nodes	2
Gateway Nodes	32
Metadata Server Nodes	2
Processor per Nodes (Pentium 4 Prestonia)	2
Total Number of Processors	1920
Processor Speed (GHz)	2.4
Theoretical Peak System Performance (TFlops)	9.2
Memory per Node (GB)	4
Total Memory (TB)	3.8
Total Local Disk Space (TB)	115
Nodes Interconnection	**Quadrics Switch**

lection because of the large I/O requests and the relatively short tracing periods. As we discuss in Section 4, I/O requests in such a large system are usually around several hundred kilobytes to several megabytes. Even in the most bursty I/O period, the total number of I/Os per node is still around tens of requests per second. Up to one hundred trace records will be generated on each node per second on average. Buffering and storing these data only has a slight impact on the system performance. Moreover, instead of tracing the whole cluster, we only study several typical scientific applications. Those applications are usually composed of two stages: the computation phase and the I/O phase. The typical I/O stage ranges from several minutes to several hours. During this period, each node usually generates several hundred kilobytes of trace data, which can be easily buffered in memory.

3.2. Applications and Traces

All of the trace data were collected from the ASCI Linux Cluster in Lawrence Livermore National Laboratory. This machine is currently in limited-access mode for science runs and file system testing. It has 960 dual-processor nodes connected through a Quadrics Switch. Two of the nodes are dedicated metadata servers and another 32 nodes are used as the gateways for accessing a global parallel file system. The detailed configuration of this machine is provided in table 1 [5]. We traced three typical parallel scientific applications during July, 2003. The total size of the traces is more than 800 megabytes.

The first application is a parallel file system benchmark, *ior2* [6], developed by LLNL. It is used for benchmarking parallel file systems using POSIX, MPIIO, or HDF5 interfaces. Basically it writes a large amount of data to one or more files and then reads them back to verify the correctness of the data. The data set is large enough to minimize the operating system caching effect. Based on different file usages, we collected three different benchmark traces, named *ior2-fileproc*, *ior2-shared*, and *ior2-stride*, respectively. All of them ran on a 512-node cluster. *ior2-fileproc* assigns an individual output file for each node, while *ior2-shared* and *ior2-stride* use a shared file for all the nodes. The difference between the last two traces is that *ior2-shared* allocates a contiguous region in the shared file for each node, while *ior2-stride* strides the blocks from different nodes into the shared file.

The second application is a physics simulation run on 343 processes. In this application, a single node gathers a large amount of data in small pieces from the others nodes. A small set of nodes then write these data to a shared file. Reads are executed from a single file independently by each node. This application has two I/O-intensive phases: the restart phase, in which read is dominant; and the result-dump phase, in which write is dominant. The corresponding traces are named *f1-restart* and *f1-write*, respectively.

The last application is another physics simulation which runs on 1620 nodes. This application use an individual output file for each node. Like the previous application, it also has a restart phase and a result-dump phase. The corresponding traces are referred as *m1-restart* and *m1-write*, respectively.

3.3. Analysis

The raw trace files required some processing before they could be easily analyzed. Some unrelated system calls and signals were filtered out. Since each node maintained its own trace records, the raw trace for each application is composed of hundreds of individual files. We merged those individual files in chronological order. Thanks to the Quadrics switch, which has a common clock, the traced time in those individual trace files was globally synchronized. Our analysis work, such as request inter-arrival time, have been greatly simplified by sorting all requests into a chronologically sorted trace file.

A good understanding of file metadata operation characteristics is important, however, our traces are not large enough to capture general metadata access patterns. Therefore, we focus more on file data I/O characterization in the following section.

4. Workload Characteristics

We present the characteristics of the workloads, including file distributions and I/O request properties. We study the distributions of file size and lifetimes and show the uniqueness of large-scale scientific workloads. We focus on three typical applications as described in Section 3.2 and examine the characteristics of I/O requests, such as the size and number of read and write requests and the burst and the distribution of I/O requests on various nodes.

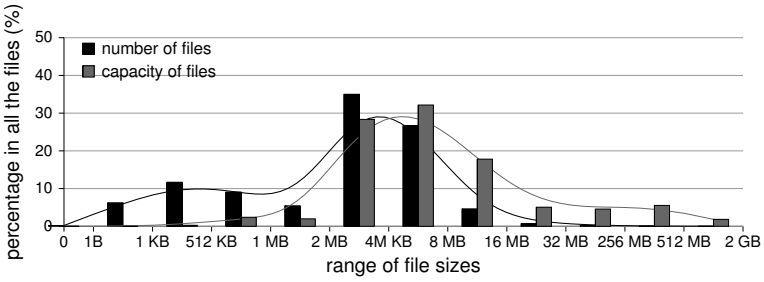
4.1. File Distributions

We collected file distributions from thirty-two file servers that were in use for the ASCI Linux cluster during the science runs phase. Each file server has storage capacity of 1.4 terabytes. The file servers were dedicated to a small number of large-scale scientific applications, which provides a good model of data storage patterns. On average, the number of files on each file server was 350,250, and each server stored 1.04 terabytes of data, more than 70% of their capacity. On most of the file servers, the number and capacity of files are similar except for five file servers. Table 2 displays statistic values of the number and capacity of files on these servers, including mean, standard deviation (std. dev.), median, minimum (min) and maximum (max).

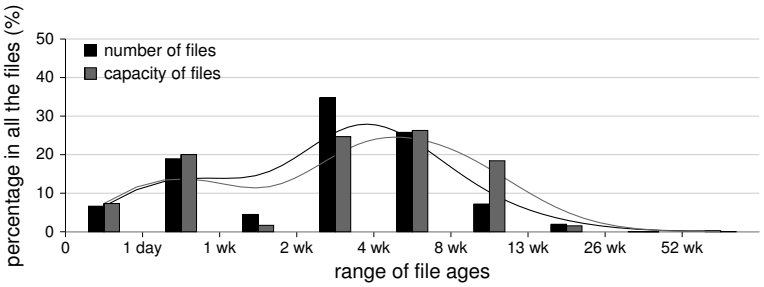
Figure 1(a) presents file size distributions by number and file capacity. The ranges of file sizes are sampled from 0–1 Byte to 1–2 Gigabytes. Some of the partitions were merged due to space limitations. We observed that over 80% of the files are between 512 kilobytes and 16 megabytes in

Table 2. File Numbers and Capacity of the 32 File Servers

	Number	Capacity
mean	305,200	1044.33 GB
standard deviation	75,760	139.66 GB
median	305,680	1072.88 GB
minimum	67,276	557.39 GB
maximum	605,230	1207.37 GB



(a) By File Sizes



(b) By File Ages

Figure 1. Distribution of Files

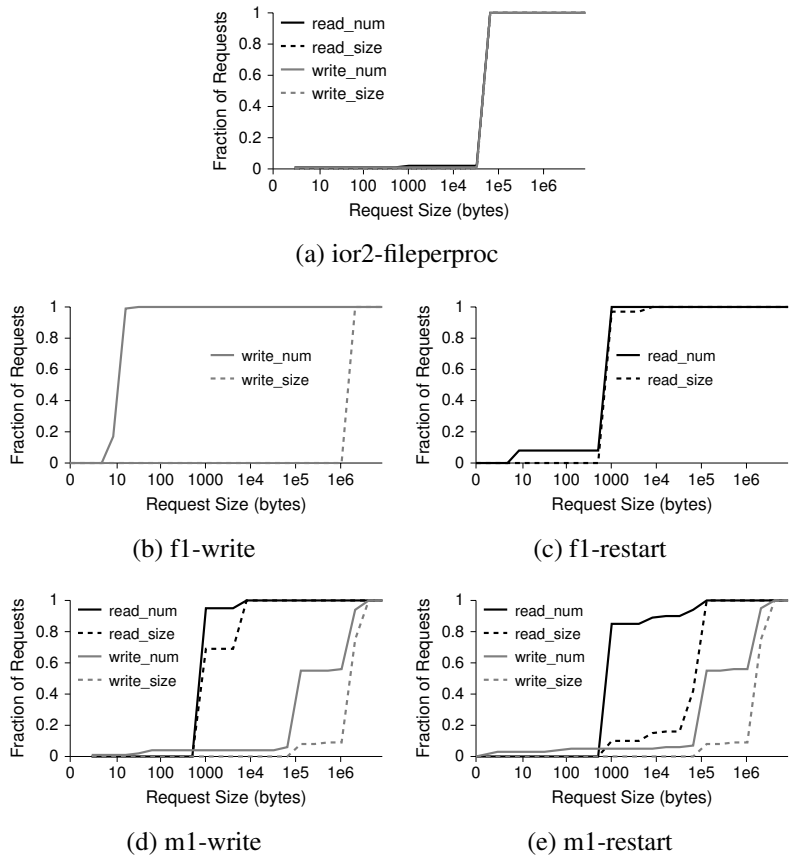
size and these files accounted for over 80% of the total capacity. Among various file size ranges, the most noticeable one is from 2 megabytes to 8 megabytes: about 61.7% of all files and 60.5% of all bytes are in this range.

We divided file lifetimes into 9 categories: from 0–1 day to 52 weeks and older. As illustrated in figure 1(b), 60% of the files and 50% of the bytes lived from 2 weeks to 8 weeks, while 6.6% of the files and 7.3% of the bytes lived less than one day. The lifetime of the traced system is about 1 year so no files lived longer than 52 weeks.

4.2. I/O Request Sizes

Figure 2 shows the cumulative distribution function of request sizes and request numbers. Since all three *ior2* benchmarks have identical request size distributions, we only show one of them. As shown in Figure 2(a), *ior2* has only an unique request size of around 64 kilobytes.

Figure 2(b) shows the write request size distribution of the result-dump stage in the physics simulation, *f1*. Almost all the write requests are smaller than 16 bytes, while almost all the I/O data are



(a) ior2-fileperproc

(b) f1-write

(c) f1-restart

(d) m1-write

(e) m1-restart

Figure 2. Cumulative Distribution Functions (CDF) of the Size and the Number of I/O Requests (X axis-logscale). The *read_num* and *write_num* curves indicate the fraction of all requests that is smaller than the size given in X axis. The *read_size* and *write_size* curves indicate the fraction of all transferred data that live in requests with size smaller than the value given in the X axis.

transferred in the requests with sizes larger than one megabyte. This turns out to be a common I/O pattern of scientific applications: a master node collects small pieces of data from all computing nodes and writes them to data files, which results in a huge number of small writes. Other nodes read and write these data files in very large chunks then. There are so few read requests in the result-dump stage and write requests in the restart stage that we actually ignore the read request curves in figure 2(b) and the write request curves in figure 2(c).

Figure 2(d) and figure 2(e) show the same write request distribution in the restart and result-dump stages of the physics simulation, *m1*. The two spikes in the *write_num* curves indicate two major write sizes: 64 kilobytes and 1.75 megabytes, respectively. Each of them accounts for 50% of all write requests. More than 95% of the data are transfered by large requests, which is also shown in Figures 2(d) and 2(e). Reads in *m1* are dominated by small requests less than 1 kilobytes. However, a small faction (less than 3%) of 8 kilobyte requests still accounts for 30% of all read data transfer. This is similar to the read distribution in Figure 2(e): only 5% of the read requests contribute to 90% of all data read.

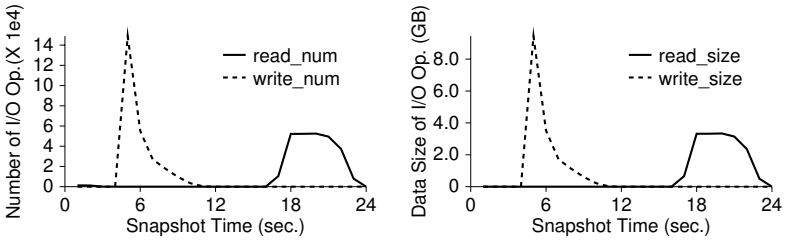
4.3. I/O Accesses Characteristics

Figure 3–5 show I/O accesses characteristics over time. The resolution for these figures is 1 second except figure 4(a), which uses a resolution of 50 seconds. Figure 3 shows that the request number distribution and the request size distribution are almost identical in *ior2* due to the fixed size requests used in those benchmarks. The *ior2-fileproc* benchmark, using the one-file-per-node model, presents the best write performance. Up to 150,000 write requests per second, totaling 9 gigabytes per second, are generated by the 512 nodes. However, the *ior2-shared* and *ior2-stride* benchmarks can only achieve 25,000 write requests per second, totaling 2 gigabytes per second. These two benchmarks use the shared-region and the shared-stride file model, respectively. We believe that the performance degradation is caused by the underlying file consistency protocol. This result is somewhat counterintuitive. The shared-region file model appears to be similar to the one-file-per-node model because the contiguous regions in the former can be analogous to the separate files in the latter. Therefore, their performance should be comparable as well. The severe performance degradation implies that the shared-file model is not optimized for this scenario.

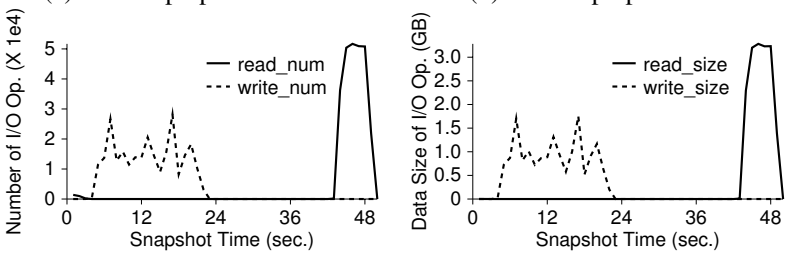
After a write, each node reads back another node's data as soon as it is available. The gaps between the write and read curves in each sub-figure reflect the actual I/O times. Obviously, the *ior2-fileproc* benchmark demonstrates much better performance: only 10 seconds are used in this model, while more than 20 seconds are needed to dump the same amount of data when using the shared file model. Since reads must be synchronous, we can easily figure out the file system read bandwidth from the *read_size* curve. The *ior2-fileproc* and *ior2-shared* benchmarks have comparable read performance. However, the *ior2-stride* has the worst read performance, which is only 100 megabytes per second for 512 nodes. This result is not surprising: the stride data layout in shared files limits the chances of large sequential reads.

Figure 4 shows the I/O access pattern of the application *f1*. As we mentioned before, *f1-write* has very few reads and *f1-restart* has very few writes. Therefore, we can ignore those requests in the corresponding figures. In Figure 4(a), we chose a resolution of 50 seconds because it becomes unreadable if we use finer time resolutions. The spike of the *write-num* curve is caused by the activities of the master node to collect small pieces of data from other computing nodes. At its peak time, nearly 1 million file system requests are issued per second. However, due to the very small request size (8 to 16 bytes), this intensive write phase contributes negligable amounts of data to the overall data size. In the rest of the application, large write requests from 48 nodes dominate the I/O activities. Requests are issued in a very bursty manner. Figure 4(b) zooms in a small region of Figure 4(a) by 1 second resolution. It shows that sharp activity spikes are separated by long idle periods. At the peak time, up to 120 megabytes per second of data are generated by 48 nodes. In the restart phase of *f1*, read requests become dominant. However, both the number and the data size of read requests are small compared to those in the write phase.

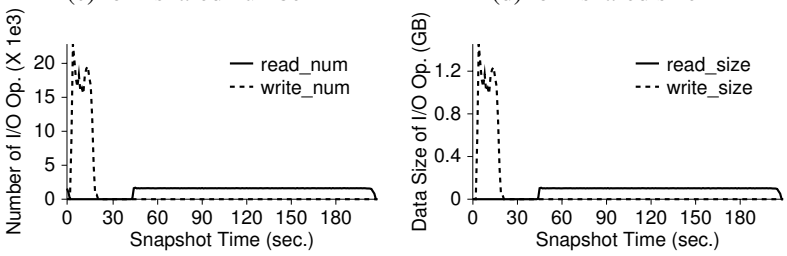
Figure 5 presents the I/O access pattern of the physics application *m1*. It demonstrates very good read performance: nearly 28 gigabytes per second bandwidth can be achieved by 1620 nodes, thanks to the large read size (1.6 megabytes – 16 megabytes). Like *f1*, its write activities are also bursty. We observed that the write curves have similar shapes in figure 5. They all begin with a sharp spike followed by several less intensive spikes. One possible explanation is that the file system buffer cache absorbs the coming write requests at the begin of the writes. However, as soon as the buffer is filled up, the I/O rate drops to what can be served by the persistent storage.



(a) ior2-fileperproc number (b) ior2-fileperproc size

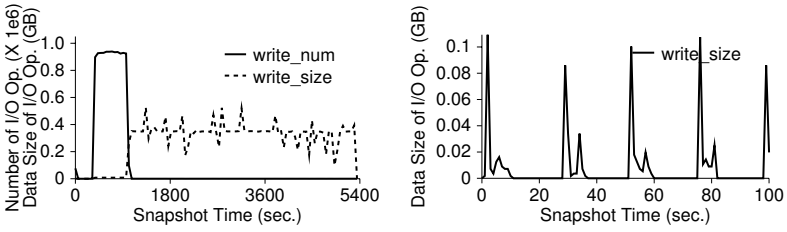


(c) ior2-shared number (d) ior2-shared size

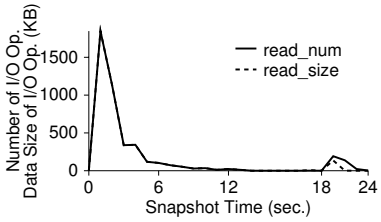


(e) ior2-stride number (f) ior2-stride size

Figure 3. I/O Requests over Time for *ior2* Benchmarks



(a) time-f1-write (b) time-f1-write-short

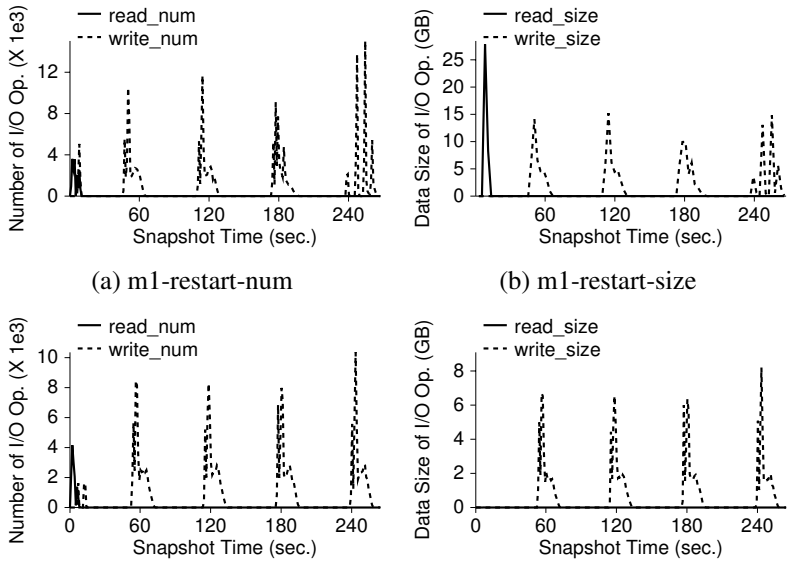


(c) time-f1-restart

Figure 4. I/O Requests over Time for *f1* Application

4.4. I/O Burstiness

To study I/O burstiness, we measure I/O request inter-arrival times. Figure 6 shows the cumulative distribution functions (CDF) of I/O request inter-arrival times. Note that the x-axis is in the logarithmic scale. Write activities are very bursty in the *ior2* benchmarks and the *f1* application: over



(a) m1-restart-num (b) m1-restart-size

(c) m1-write-num (d) m1-write-size

Figure 5. I/O Requests over Time for *m1* Application

65–100% of write requests have inter-arrival times within 1 millisecond. In *ior2* and *f1*, most of the write activities are due to memory dump and I/O nodes can issue write requests quickly. However, write activities on *m1* are less intensive than those on *ior2* and *f1*

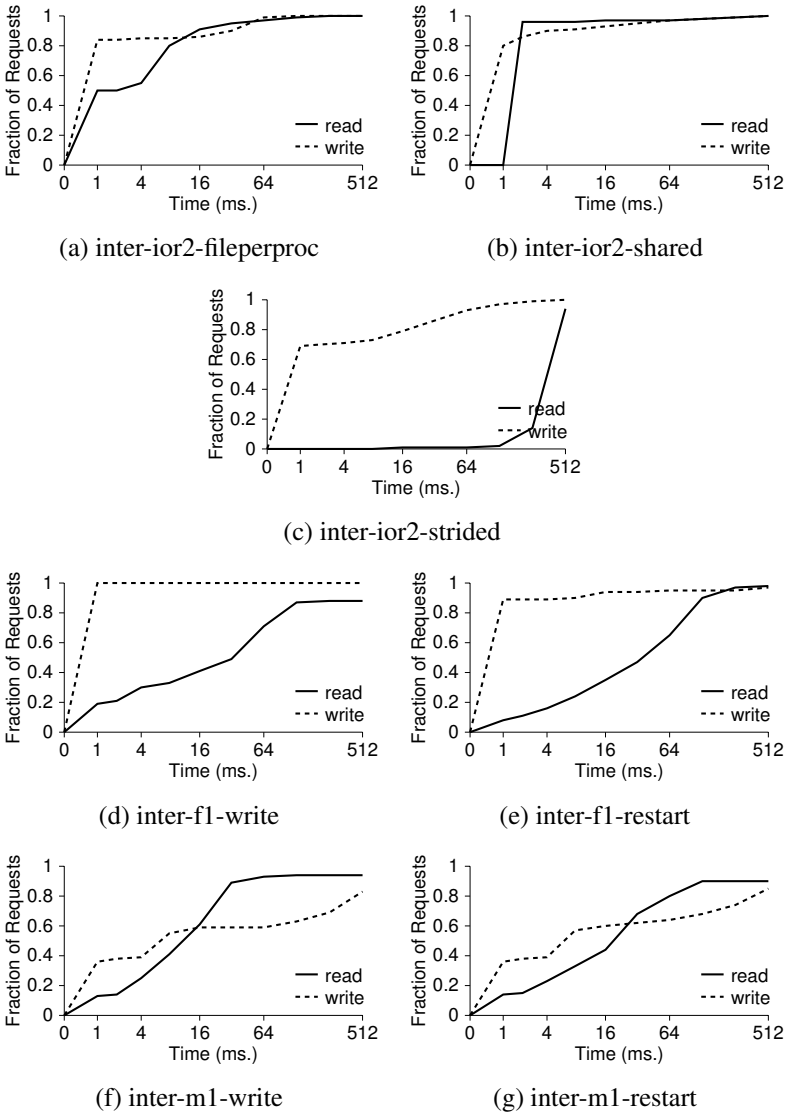
On the other hand, read requests are generally less intensive than write requests because reads are synchronous. In particular, Figure 6(c) indicates that *ior2* under shared-strided files suffers low read performance, as described in Section 4.3. In this scenario, data are interleaved in the shared file and read accesses are not sequential.

4.5. I/O Nodes

In this section, we study the distributions of I/O request sizes and numbers over nodes, as shown in Figure 7. For the *ior2* benchmarks, read and writes are distributed evenly among nodes, as shown in Figures 7(a) and 7(b), because each node executes the same sequence of operations in these benchmarks.

In the physics application *f1*, a small set of nodes write gathered simulated data to a shared file. Therefore, only a few nodes have significant I/O activity in their write phase and most of the transfered data are from large write requests (14% of the write requests), as shown in Figures 7(c) and 7(d). There is little read activity in the write phase. However, read requests are evenly distributed among nodes in the restart phase and their sizes are around 1 megabyte, as shown in Figures 7(e) and 7(f). There is little write activity in the restart phase.

In the restart and write phases of the physics application *m1*, I/O activity is well balanced among nodes, as shown in Figures 7(g)–7(j). We also observe significant write activity in the restart phase.



(a) inter-ior2-fileperproc (b) inter-ior2-shared

(c) inter-ior2-strided

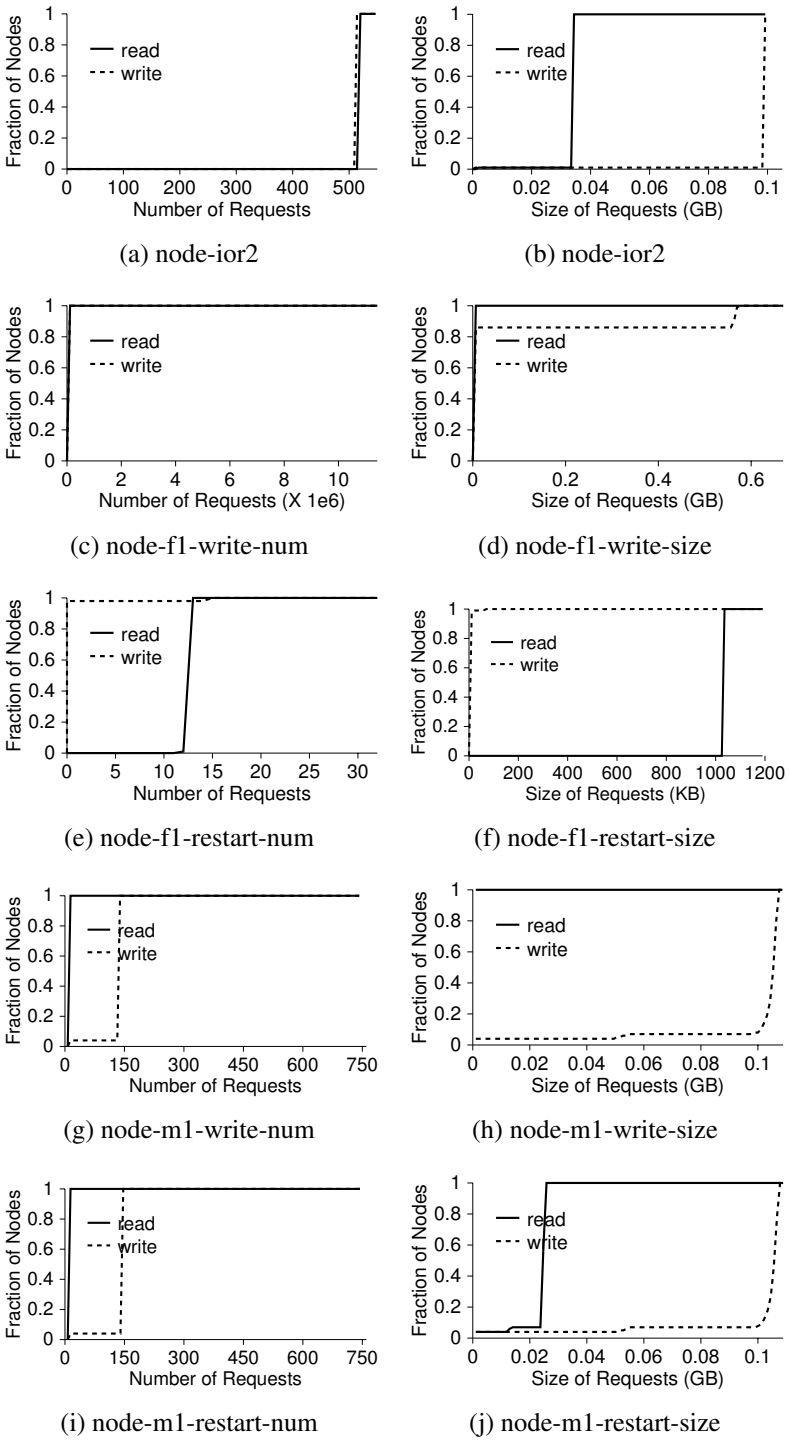
(d) inter-f1-write (e) inter-f1-restart

(f) inter-m1-write (g) inter-m1-restart

Figure 6. Cumulative Distribution Functions (CDF) of Inter-arrival Time of I/O Requests (X axis-logscale)

Table 3. File Open Statistics

Applicatons	Overall Number of File Opens			Number of Data File Opens		
	Read/Write	Read	Write	Read/Write	Read	Write
ior2	6,656	5,121	0	1,024	0	0
f1-write	3,871	6,870	718	98	10	34
f1-restart	3,773	6,179	0	0	343	0
m1-restart	17,824	22,681	12,940	0	1,620	12,960
m1-write	17,824	21,061	12,960	0	0	12,960



(a) node-ior2 (b) node-ior2

(c) node-f1-write-num (d) node-f1-write-size

(e) node-f1-restart-num (f) node-f1-restart-size

(g) node-m1-write-num (h) node-m1-write-size

(i) node-m1-restart-num (j) node-m1-restart-size

Figure 7. Cumulative Distribution Functions (CDF) of the Size of I/O Requests over Nodes

Table 4. Operations During File Open

Applications	Avg. open time		Avg. IOs per Open		Avg. IO Size per Open	
	Overall	Data File	Overall	Data File	Overall	Data File
ior2-fileproc	0.4 sec	4.5 sec	44.4	512.0	2.8 MB	32.8 MB
ior2-shared	0.7 sec	5.2 sec	44.4	512.0	2.8 MB	32.8 MB
ior2-stride	7.6 sec	26.57 sec	44.4	512.0	2.8 MB	32.8 MB
f1-write	20.2 sec	504.9 sec	14.8	142161	2.4 MB	3993.5 MB
f1-restart	0.02 sec	0.1 sec	0.5	1	<< 1 MB	<< 1 MB
m1-restart	1.2 sec	3.9 sec	4.2	15.3	3.7 MB	8.5 MB
m1-write	1.2 sec	2.4 sec	4.3	17	3.1 MB	6.5 MB

4.6. File Opens

In this section, we study the file open patterns of the applications. We use the term *data files* to refer to those files that actually store results dumped from applications.

In all applications, files tend to be opened as read/write or read-only. We only observe significant write-only files in the physics application *m1*, as shown in table 3. However, the data files are opened either read-only or write-only except for the benchmark *ior2*. The open operations on the data files only account for a small portion of the overall files opened. Given the fact that the data file operations dominate the overall I/O, the small number of data file opens implies longer open time and more I/O operations during each open. As listed in table 4, the open duration of data files ranges from several seconds to several hundred seconds, which is typically 2 to 20 times longer than overall file open durations. The average number of operations and the size of data files on each open operation are also much larger than those on the overall files. For example, up to 400 MB data are transferred during each data file open in physical application *f1-write*.

5. Conclusion

In this study, we analyze application traces from a cluster with hundreds of processing nodes. On average, each application has only one or two typical request sizes. Large requests from several hundred kilobytes to several megabytes are very common. Although in some applications, small requests account for more than 90% of all requests, almost all of the I/O data are transferred by large requests. All of these applications show bursty access patterns. More than 65% of write requests have inter-arrival times within one millisecond in most applications. By running the same benchmark on different file models, we also find that the write throughput of using an individual output file for each node exceeds that of using a shared file for all nodes by a factor of 5. This indicates that current file systems are not well optimized for file sharing. In all those applications, almost all I/Os are performed on a small set of files containing the intermediate or final computation results. Such files tend to be opened for a relatively long time, from several seconds to several hundred seconds. And a large amount of data are transferred during each open.

Acknowledgments

Feng Wang, Qin Xin, Scott Brandt, Ethan Miller, Darrell Long were supported in part by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory under contract B520714. Bo Hong was supported in part by the National Science Foundation under grant number CCR-073509. Tyce McLarty's effort was under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48. This document was reviewed and released as unclassified with unlimited distribution as LLNL-UCRL-CONF-201895.

We are also grateful to our sponsors: National Science Foundation, USENIX Association, Hewlett Packard Laboratories, IBM Research, Intel Corporation, Microsoft Research, ONStor, Overland Storage, and Veritas.

References

[1] S. J. Baylor and C. E. Wu. Parallel I/O workload characteristics using Vesta. In *Proceedings of the IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems (IOPADS '95)*, pages 16–29, Apr. 1995.

[2] DOE National Nuclear Security Administration and the DOE National Security Agency. Proposed statement of work: SGS file system, Apr. 2001.

[3] D. Kotz and R. Jain. I/O in parallel and distributed systems. In A. Kent and J. G. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 40, pages 141–154. Marcel Dekker, Inc., 1999. Supplement 25.

[4] D. F. Kotz and N. Nieuwejaar. File-system workload on a scientific multiprocessor. *IEEE Parallel and Distributed Technology*, 3(1):51–60, 1995.

[5] Lawrence Livermore National Laboratory. ASCI linux cluster. http://www.llnl.gov/linux/alc/, 2003.

[6] Lawrence Livermore National Laboratory. IOR software. http://www.llnl.gov/icc/lc/siop/downloads/download.html, 2003.

[7] E. L. Miller and R. H. Katz. Input/output behavior of supercomputing applications. In *Proceedings of Supercomputing '91*, pages 567–576, Nov. 1991.

[8] A. L. Narasimha Reddy and P. Banerjee. A study of I/O behavior of perfect benchmarks on a multiprocessor. In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 312–321. IEEE, 1990.

[9] B. K. Pasquale and G. C. Polyzos. A static analysis of I/O characteristics of scientific applications in a production workload. In *Proceedings of Supercomputing '93*, pages 388–397, Portland, OR, 1993. IEEE.

[10] B. K. Pasquale and G. C. Polyzos. Dynamic I/O characterization of I/O-intensive scientific applications. In *Proceedings of Supercomputing '94*, pages 660–669. IEEE, 1994.

[11] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best. Characterizing parallel file-access patterns on a large-scale multiprocessor. In *Proceedings of the 9th International Parallel Processing Symposium (IPPS '95)*, pages 165–172. IEEE Computer Society Press, 1995.

[12] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, July 2003.

[13] E. Smirni, R. A. Aydt, A. A. Chien, and D. A. Reed. I/O requirements of scientific applications: An evolutionary view. In *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 49–59. IEEE, 1996.

[14] E. Smirni and D. Reed. Lessons from characterizing the input/output behavior of parallel scientific applications. *Performance Evaluation: An International Journal*, 33(1):27–44, June 1998.

V:Drive - Costs and Benefits of an Out-of-Band Storage Virtualization System *

André Brinkmann, Michael Heidebuer, Friedhelm Meyer auf der Heide,
Ulrich Rückert, Kay Salzwedel, and Mario Vodisek
Paderborn University

Abstract

The advances in network technology and the growth of the Internet together with upcoming new applications like peer-to-peer (P2P) networks have led to an exponential growth of the stored data volume. The key to manage this data explosion seems to be the consolidation of storage systems inside storage area networks (SANs) and the use of a storage virtualization solution that is able to abstract from the underlying physical storage system.

In this paper we present the first measurements on an out-of-band storage virtualization system and investigate its performance and scalability compared to a plain SAN. We show in general that a carefully designed out-of-band solution has only a very minor impact on the CPU usage in the connected servers and that the metadata management can be efficiently handled. Furthermore we show that the use of an adaptive data placement scheme in our virtualization solution V:Drive can significantly enhance the throughput of the storage systems, especially in environments with random access schemes.

1. Introduction

The advances in networking technology and the growth of the Internet have enabled and accelerated the emergence of new storage consuming applications like peer-to-peer (P2P) networking, video-on-demand, and data warehousing. The resulting exponential growth of the stored data volume requires a new storage architecture, while the management of the traditional, distributed *direct attached storage* (DAS) architecture has shown to be intractable from a business perspective. The first step towards this new storage architecture is the consolidation of the servers and storage devices inside a *storage area network* (SAN). In a SAN, the formerly fixed connections between storage and servers are broken-up and both are attached to the high-speed dedicated storage network. The introduction of a storage area network can significantly improve the reliability, availability, manageability, and performance of servers and storage systems.

Nevertheless, it has been shown that the potential of a SAN can only be fully exploited with the assistance of storage management software, and here particulary with the help of a virtualization system. Storage virtualization is often seen as the key technology in the area of storage management. But what actually is storage virtualization? A good definition has been given by the Storage Networking Industry Association SNIA [8]:

"[Storage virtualization is] an abstraction of storage that separates the host view [from the] storage system implementation."

This abstraction includes the physical location of a data block as well as the path from the host to the storage subsystem through the SAN. Therefore, it is not necessary that the administrator of a SAN is aware of the distribution of data elements among the connected storage systems. Generally, the administrator only creates a virtual volume and assigns it to a pool of physical volumes, where each physical volume can be of different size. Then, a file system or a database can work upon this virtual volume and the virtualization software provides a consistent allocation of data elements on the storage systems. It is even possible that a large number of virtual volumes share a common storage pool.

The use of a virtualization environment has many advantages compared to the traditional approach of assigning an address space to a fixed partition. The most obvious one is that a virtual disk can become much larger than the size of a single disk or even than a single RAID-system [7]. When using virtualization software, the size of a virtual disk is only limited by the restrictions inherent to the operating system and the total amount of available disk capacity.

Another important feature of virtualization software is a much better utilization of disk capacity. It has been shown

*Partially supported by the DFG Transferbereich 40 and the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

that in the traditional storage model only 50% of the available disk space is used. The disk utilization can be increased up to 80% through the central and more flexible administration of virtualization software. Thus, the required storage capacity and, with it, the hardware costs of a storage area network can be reduced significantly. Furthermore, virtualization software offers new degrees of flexibility. Storage systems can be added to or removed from storage pools without downtime, thus enabling a fast adaptation to new requirements. These storage systems do not have to be from a single vendor, so that the traditional vendor-locking of customers can be avoided.

Virtualization software can be implemented as out-of-band virtualization or in-band virtualization, inside the storage subsystems, or as logical volume manager (LVM) inside the hosts. In an out-of-band virtualization system, the virtualization is done inside the kernel of the hosts and all participating hosts are coordinated by one or more additional SAN appliance. In this paper we will focus on the analysis of our out-of-band solution *V:Drive*.

Chapter 2 of this paper introduces the design of *V:Drive*. In chapter 3 we present the first measurements on an out-of-band storage virtualization system and investigate its performance and scalability compared to a plain SAN. We show that a carefully designed out-of-band solution has only a very minor impact on the CPU usage in the connected hosts and that the metadata management can be efficiently implemented. Furthermore we give evidence that the use of an adaptive data placement scheme can significantly enhance the throughput of storage systems, especially in environments with random access schemes.

2. *V:Drive* Design

In this chapter we will describe the design of our out-of-band virtualization solution *V:Drive*. From the architectural perspective, *V:Drive* consists of a number of cooperating components: one or more SAN appliances which are responsible for the metadata management and the coordination of the hosts (see section 2.2), the virtualization engine inside the kernel of the hosts (see section 2.3), and a graphical user interface (GUI).

From a logical point of view, *V:Drive* offers the ability to cluster the connected storage devices into storage pools that can be combined according to their age, speed, or protection against failures. Each storage pool has its own storage management policy describing individual aspects like logical and physical block size or redundancy constraints. A large number of virtual volumes can share the capacity of a single storage pool.

The capacity of each disk in a storage pool is partitioned into minimum sized units of contiguous data blocks, so called *extents*. The extent size need not be constant inside a storage pool and can change over time. In general, smaller extents can guarantee better load balancing, while bigger extents result in a smaller management overhead and less disk head movements in case of sequential accesses. The extents are distributed among the storage devices according to the *Share* strategy which is able to guarantee an almost optimal distribution of the data blocks across all participating disks in a storage pool (see Section 2.1).

2.1. The Share-Strategy

Any virtualization strategy depends on the underlying data distribution strategy. Such a distribution is challenging if the system is allowed to contain heterogeneous storage components. The main task of a distribution strategy is an even distribution of data blocks and an even distribution of requests among the storage devices. Therefore, it has a strong impact on the scalability and the performance of the SAN. It can be shown, that a static data placement scheme is generally not able to fulfill the given requirements.

We have developed a new adaptive distribution scheme that has been implemented in *V:Drive*, called *Share*-strategy [2]. In this paper we will present *Share* without data replication. Of course it is possible to support replication inside *Share*, e.g. by a scheme proposed in [4]. For other static and dynamic placement schemes, see [6, 3, 4, 5].

Share works in two phases. In the first phase, the algorithm reduces the problem of mapping extents to heterogeneous disks to a number of homogeneous ones. The result is a number of volumes which are equally likely to store the requested extent. In the second phase, we use any distribution strategy that is able to map extents to equal sized disks (see e.g. [6]).

The reduction phase is based on two hash functions $h : \{1, ..., M\} \to [0, 1)$ and $g : \{1, ..., N\} \to [0, 1)$ where M is the maximal number of extents in the system and N is the maximal number of disks that are allowed to participate, respectively.

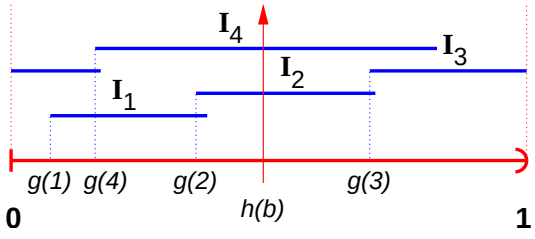


Figure 1. *Hashing scheme in Share*

The reduction phase works as follows: Initially or after every change in the system configuration, we map the

starting points of sub-intervals of certain length into a [0,1) interval using the hash function *g*. The length of these sub-intervals I_i corresponds to the capacity d_i of disk *i*. To ensure that the whole interval is covered by at least one sub-interval we need to stretch each of the sub-intervals by a factor *s*. In other words, the sub-interval I_i starts at $g(i)$ and ends at $(g(i) + s \cdot d_i) \bmod 1$.

The extents are hashed into the same interval using *h* where the quality of *h* ensures an even distribution of all extents over the whole interval. Now, an extent can be accessed by calculating its hash value and then deriving all sub-intervals that value falls into. Any efficient uniform strategy can be applied to get the correct disk out of the number of possible candidates. It can be shown that the fraction of extents stored on a disk and the number of requests to a disk are proportional to its capacity and that the number of extent replacements in case of any change in the number or kind of disks is nearly minimal (see [2] for more detail).

2.2. SAN Appliance and Metadata Management

To ensure a consistent view of the SAN and a proper configuration of the hosts and storage devices, one or more SAN appliances are connected to the SAN. The appliances keep track of all necessary metadata structures. These metadata include among others the partitioning of the storage devices into storage pools and virtual volumes, access rights of the hosts, and the allocation of extents on the storage devices.

The metadata appliance consists of a number of separate modules which are arranged around the *V:Drive* database, including the *Disk-Agent*, the *Host-Interface*, the *Disk-Manager*, and the *Administration Interface*. The interface to the database is standard SQL, implemented in many commercial and free databases. All components of the appliance can be executed on a single machine or can run in a distributed fashion.

Information about the state of the SAN are collected by the *Disk-Agent* that is responsible for detecting disk partitions and for finding changes in the system configuration. Each newly detected suitable partition is labelled with a unique ID and is made available by updating the database.

The *Host-Interface* is connected to the servers via Ethernet/IP. A data transfer between a server and the host interface is issued if the configuration of the SAN has been changed, if the server has started and has to load its configuration, or if the server accesses a virtual address for the first time and has to allocate an extent on the corresponding disk.

If the configuration of the storage system changes, a small number of extents has to be redistributed in order to guarantee close to optimal performance. The *Disk-Manager* is responsible for this redistribution tasks. After each change of a storage pool it checks each allocated extent if it has to be relocated. In such a case, the extent is moved online to its new location in a way that ensures the consistency of the data before, during, and after the replacement process.

The administrator can access the metadata via the graphical user interface. The administration interface contains all the necessary functionality to manage enterprise wide storage networks: administration of storage systems, storage pools, and virtual devices, authentication and authorization, security, and statistics.

2.3. Kernel Integration

The host software basically consists of a kernel module which is linked to the operating system of the participating servers and some additional applications running in the user space. Currently, modules for the Linux kernel 2.2 and 2.4 are available.

If a data block needs to be read from a virtual disk, the file system generates a block I/O request and passes it to the kernel module where it is processed and transmitted to the appropriate physical disk. To perform the transformation from a virtual address to a physical address, the kernel keeps all necessary information, like existing storage pools, assignments of virtual and physical disks to the pools, storage policies etc. These information are given to the kernel initially or on-demand by the metadata server.

3. Results

In this section we will present the experimental results of our virtualization approach. The test system consists of two Pentium servers connected to an FC-AL array with 8 fibre channel disks. Both servers have 2 Pentium II processors with 450 MHz and 512 kilobyte cache. Furthermore, they have local access to a mirrored disk drive containing the operating system and all relevant management information. Both servers run a Linux 2.4.18 kernel (Red Hat) and use gcc version 3.2.2 as the C compiler. The access to the disks is enabled by a QLogic qla2300 host bus adapter. The FC-AL array consists of four 17 Gigabyte and four 35 Gigabyte fibre channel disks. They are connected with the server via an 1 Gigabit switch. Each disk is partitioned into one partition covering the whole disk.

For stressing the underlying I/O subsystem we used the Bonnie file system benchmark [1]. We changed the original source code such that we could derive more information concerning the overhead of our solution. The simple design and easy handling of Bonnie makes it a suitable tool for testing I/O performance. It performs, among others, the following operations on a number of file of desired size: it

reads and writes the random content of each character of the file separately, it reads and writes each block of the file, and it concurrently accesses arbitrary blocks in the file.

The first two tests access a number of data files sequentially. Such a scenario is rather unlikely in practice but it is able to give an idea of the maximal performance of the I/O subsystem. More suited to model real world scenarios is the last test, because we have to access arbitrary blocks in some files. We set the overall file size to 4 GB (4 times the size of the main memory) to reduce caching effects. The size of the extents was fixed to 1 MB.

To derive the overhead of our approach we compare our approach to the performance of a plain disk (labeled with the device name, e.g. SDA). More specific, we investigate the influence of each component of our solution to the overall throughput. For that we distinguish the following cases:

1. Clean System (C): Nothing is known in advance.
2. Transfer (T): All extents exist in the database and have only to be transferred to the driver.
3. Driver (D): The driver has all information locally and does only perform the mapping of addresses.

The number in parentheses behind the letters C,T,D in the charts axes is equivalent to the number of physical volumes belonging to the corresponding storage pool. If not mentioned otherwise, the storage pool consists of a single physical volume.

Throughout the experiments the CPU usage for our approach was indistinguishable from the CPU usage when accessing the plain disk. Due to space limitations the corresponding figures are omitted.

3.1. Impact of Extent Requests

Figure 2 shows the throughput for the different settings when each character is written separately. Note, that we only get an overhead when the extent is accessed for the first time. Therefore, the induced costs are credited to many data accesses and their effect becomes marginal. The differences are mostly due to cache effects.

The situation is very different when it comes to blockwise accesses. Here, the fraction of block accesses which induce overhead is much higher. Figure 3 shows the performance not only for the different settings but also for varying sizes of corresponding storage pools. Surprisingly, we lose roughly 40% when using only one disk. The reason for that lies mostly in the special sequential access pattern. The achieved high throughput could only be gained because the layout of the data blocks on disk enables a sweep of the disk head, minimizing the head movements. Modern file systems take that into account and adapt their data layout accordingly. But we destroy the careful layout because we

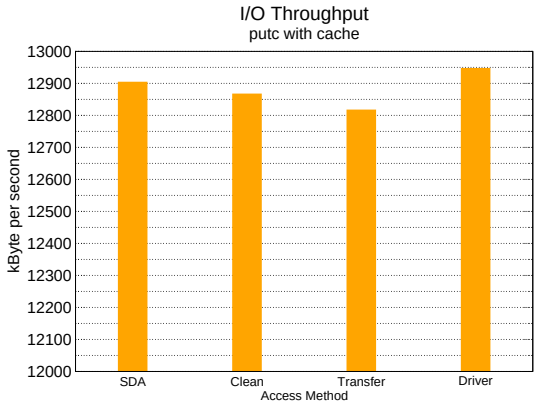


Figure 2. Comparison of the sequential output per character.

access the data in extents instead of data blocks. When allocating an extent the metadata server returns the first free position on the disk that is big enough to host the extent. Therefore, a sequential access of the file system results in higher movement of the disk head and only the sustained throughput of a disk could be achieved.

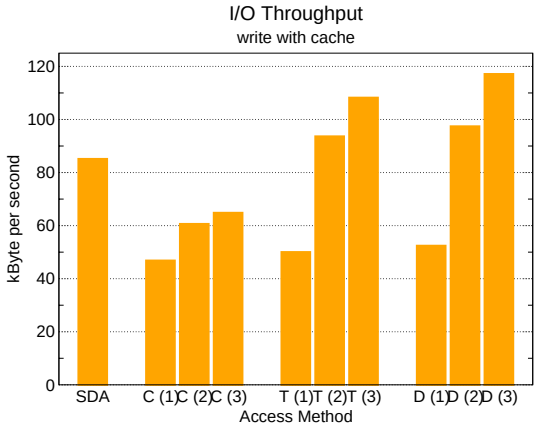


Figure 3. Comparison of the sequential output per block.

Surprisingly, this effect is compensated by parallel accesses when feeding the virtual device from more than one disk. Due to the fact that the operating system issues the write requests to the main memory and returns immediately, we achieve parallelism and get roughly the sustained performance of two disks. We could top the performance significantly, even if the access pattern does not allow for much parallelism. This indicates that the overhead induced by the driver alone is not a limiting factor. Only a clean sys-

tem with many extent allocations is not able to use many disks to increase the performance compared to a single disk. But in a real-world application a data block is accessed many times and the overhead occurs only once.

3.2. Block Read Performance

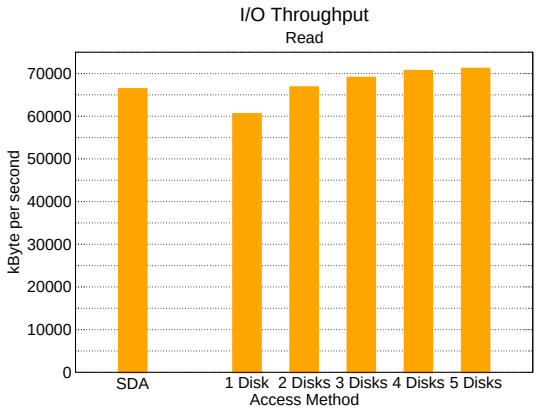


Figure 4. Comparison of performance of block accesses.

The read access is different from a write because it has to wait until the data is delivered from the disk. This gives the operating system enough time to rearrange a larger number of data accesses and, hence, accesses the disk in a sweeping manner. Figure 4 gives evidence for that. We lose only about 9% compared to the performance of a plain disk. As noted above the access pattern allows little parallelism. Hence, the increasing of the number of disks has only a small impact on the overall performance.

3.3. Random Seeks

To get the number of random seeks per second Bonnie creates 3 threads performing the data requests. It is our opinion that this test is closest to practice because on a storage server there are different application generating rather unpredictable block accesses. Figure 5 compares the number of seeks per second for all approaches. Again, the overhead induced by the *V:Drive* solution is too small to measure once the extents are allocated.

Note, that the impact of more disks decreases the more disk participate in the storage pool. This is due to the fact that the number of scheduled requests stays constant. That means, that the likelihood of parallel accesses to all disks decreases with the number of disks. If we would access the storage pool with more virtual devices the scaling would be much better.

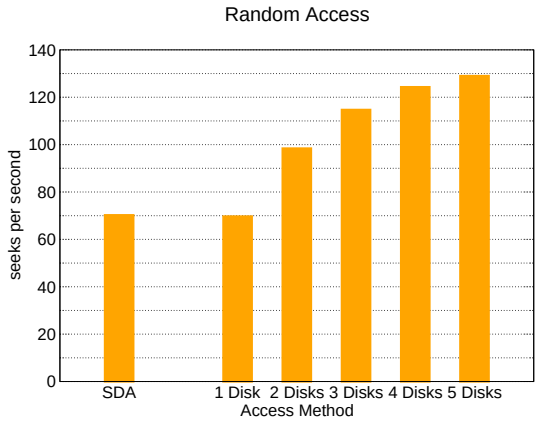


Figure 5. Comparison of the performed number of seeks per second.

4. Conclusion

In this paper we presented a virtualization environment that is based on the randomized Share-strategy. The shown results give evidence that such an approach is not only feasible but also efficient. Especially the performance of random seeks to files via Bonnie hints that *V:Drive* scales nicely with a growing storage network.

References

[1] T. Bray. Bonnie source code. http://www.textuality.com.

[2] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, Adaptive Placement Schemes for Non-Uniform Distribution Requirements. In *Proceedings of the 14th ACM SPAA Conference*, 2002.

[3] T. Cortes and J. Labarta. Extending Heterogeneity to RAID level 5. In *Proceedings of the USENIX Annual Technical Conference*, 2001.

[4] R. J. Honicky and E. L. Miller. A Fast Algorithm for Online Placement and Reorganization of Replicated Data. In *Proceedings of the 17th IPDPS Conference*, 2003.

[5] R. J. Honicky and E. L. Miller. Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution. In *Proceedings of the 18th IPDPS Conference*, 2004.

[6] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *In Proceeding of the 29th ACM STOC Conference*, pages 654–663, 1997.

[7] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, 1988.

[8] The Storage Networking Industry Association (SNIA). Storage Virtualization I: What, Why, Where and How.

Identifying Stable File Access Patterns

Purvi Shah
University of Houston
purvi@cs.uh.edu

Jehan-François Pâris[1]
University of Houston
paris@cs.uh.edu

Ahmed Amer[2]
University of Pittsburgh
amer@cs.pitt.edu

Darrell D. E. Long[3]
U. C. Santa Cruz
darrell@cs.ucsc.edu

1. Introduction

Disk access times have not kept pace with the evolution of disk capacities, CPU speeds and main memory sizes. They have only improved by a factor of 3 to 4 in the last 25 years whereas other system components have almost doubled their performance every other year. As a result, disk latency has an increasingly negative impact on the overall performance of many computer applications.

Two main techniques can be used to mitigate this problem, namely *caching* and *prefetching*. Caching keeps in memory the data that are the most likely to be used again while prefetching attempts to bring data in memory before they are needed. Both techniques are widely implemented at the data block level. More recent work has focused on caching and prefetching entire files.

There are two ways to implement file prefetching. *Predictive prefetching* attempts to predict which files are likely to be accessed next in order to read them before they are needed. While being conceptually simple, the approach has two important shortcomings. First, the prefetching workload will get in the way of the regular disk workload. Second, it is difficult to predict file accesses sufficiently ahead of time to ensure that the predicted files can be brought into main memory before they are needed.

A more promising alternative is to group together on the disk drive files that are often accessed at the same time [3]. This technique is known as *implicit prefetching* and suffers none of the shortcomings of predictive prefetching because each cluster of files can now be brought into main memory in a single I/O operation. The sole drawback of this new approach is the need to identify stable file access patterns in order to build long-lived clusters of related files.

We present here a new file predictor that identifies stable access patterns and can predict between 50 and 70 percent of next file accesses over a period of one year. Our *First Stable Successor* keeps track of the successor of each individual file. Once it has detected *m* successive accesses to file *Y*, each immediately following an access to file *X*, it predicts that file *Y* will always be the successor of file *X* and never alters this prediction.

The remainder of this paper is organized as follows. Section 2 reviews previous work on file access prediction. Section 3 introduces our *First Stable Successor* predictor and Section 4 discusses its performance. Finally, Section 5 states our conclusions

2. Previous Work

Palmer *et al.* [8] used an associative memory to recognize access patterns within a context over time. Their predictive cache, named *Fido*, learns file access patterns within isolated access *contexts*. Griffioen and Appleton presented in 1994 a file prefetching scheme relying on graph-based relationships [4]. Shriver *et al.* [10] proposed an analytical performance model to study the effects of prefetching for file system reads.

Tait and Duchamp [11] investigated a client-side cache management technique used for detecting file access patterns and for exploiting them to prefetch files from servers. Lei and Duchamp [6] later extended this approach and introduced the *Last Successor* predictor. More recent work by Kroeger and Long introduced more effective schemes based on context modeling and data compression [5].

Two much simpler predictors, *Stable Successor* (or Noah) [1] and *Recent Popularity* [2], have been recently proposed. The *Stable Successor* predictor is a refinement of the Last Successor predictor that attempts to filter out noise in the observed file reference stream. Stable Successor keeps track of the last observed successor of every file, but it does not update its past prediction of the successor of file *X* before having observed *m* successive instances of file *Y* immediately following instances of file *X*. Hence, given the sequence:

S: ABABABACABACABADADADA

Stable Successor with $m = 3$ will first predict that *B* is the successor of *A* and will not update its prediction until it encounters three consecutive instances of file *D* immediately following instances of file *A*.

The *Recent Popularity* or *k-out-of-n* predictor maintains the *n* most recently observed successors of each file. When attempting to make a prediction for a given file, Recent Popularity searches for the most

[1] Supported in part by the National Science Foundation under grant CCR-9988390.
[2] Supported in part by the National Science Foundation under grant ANI-0325353.
[3] Supported in part by the National Science Foundation under grant CCR-0204358.

popular successor from the list. If the most popular successor occurs at least k times then it is submitted as a prediction. When more than one file satisfies the criterion, recency is used as the tiebreaker.

3. The First Stable Successor Predictor

All the predictors are dynamic in the sense that they reflect changes in file access patterns and modify accordingly their predictions. The sole existing static predictor is *First Successor* [1], which always predicts the first encountered successor of file *X* as its successor. It is a rather crude predictor and was found to perform much worse than all Last Successor, Stable Successor or Recent Popularity.

There are two explanations for this poor performance. First, First Successor cannot reflect changes in file access patterns. Second, it bases all its predictions on a single observation.

As shown on Figure 1, the *First Stable Successor* (FSS) predictor remedies this second limitation by requiring m successive instances of file *Y* immediately following instances of file *X* before predicting that file *Y* is the successor of file *X*. Otherwise it makes no prediction. When $m = 1$, the FSS predictor becomes identical to the First Successor protocol and predicts that that file *Y* is the successor of file *X* once it has encountered a single access to file *Y* immediately following an access to file *X*.

A large value of m will result into fewer predictions than a smaller value of m but will also increase the likelihood that these predictions will be correct. This provides us with a relatively easy way to tune the protocol by either increasing m whenever we want to reduce the number of false predictions or decreasing it whenever we want to increase the total number of predictions.

4. Performance Evaluation

When comparing the effectiveness of file predictors, one is often confronted with two primary metrics, *success-per-reference* and *success-per-prediction*. Given the dependent nature of these metrics, it is impossible to use either of them alone when assessing the performance of any given predictor. For example, a predictor that has a 99% *success-per-prediction* rate would be considered impractical if it could only be used on 5% of the references. Conversely, predictors that have a high *success-per-reference* rate may also give rise to a high number of incorrect predictions that may tax the file system to the extent that it outweighs any improvements due to predictive prefetching.

We will use a third metric integrating both aspects of the predictor performance. Consider first the two possible outcomes of an incorrect prediction. If we assume no preemption, the next file access will have to wait while the predicted file is loaded into the cache. The

Assumptions:

G is file being currently accessed
F its direct predecessor
FirstStableSuccessor(*F*) is last prediction made for the successor of *F*
LastSuccessor(*F*) is last observed successor of *F*
Count(*F*) is a counter
m is minimum number of consecutive identical successors to declare a First Stable Sucessor

Algorithm:

```
if FirstStableSuccessor(F) is undefined then
        if LastSuccessor(F) = G then
                Counter(F) ← Counter(F) + 1
        else
                Counter(F) ← 1
        end if
        if Counter(F) = m then
                FirstStableSuccessor(F) ←G
        end if
end if
```

Figure 1 The First Stable Successor Predictor

cost of the incorrect prediction is thus one additional cache miss. Allowing preemption would reduce this delay and decrease the penalty. Note that the incorrect prediction will have no other adverse effect on the cache performance as long as the cache replacement policy expels first the files that were never accessed.

We define the *effective success rate per reference* of a predictor as the ratio:

$$\frac{N_{corr} - \alpha N_{incorr}}{N_{ref}}$$

where N_{corr} is the number of correct predictions, N_{incorr} the number of incorrect predictions and N_{ref} the number of references and the α factor represents the impact of file fetch preemption on the performance of the predictor. A zero value for α corresponds to the situation where incorrect predictions incur no cost because all predicted file fetches can be preempted when found to be incorrect without any further delay. A unit value assumes that there is no fetch preemption, and all ongoing fetches must be completed, whether correctly predicted or not. An intermediate α value corresponds to situations where preemption is possible, but at some cost less than the cost of a file fetch. Computing the *effective success rate per reference* for α values of, say, 0.0, 0.5 and 1.0 will permit us to compare predictors for a realistic range of file-system implementations.

We evaluated the performance of our FSS predictor by simulating its operation on two sets of file traces. The first set consisted of four file traces collected using Carnegie Mellon University's *DFSTrace* system [7]. The traces include *mozart,* a personal workstation, *ives*, a system with the largest number of users, *dvorak*, a system with the largest proportion of write activity,

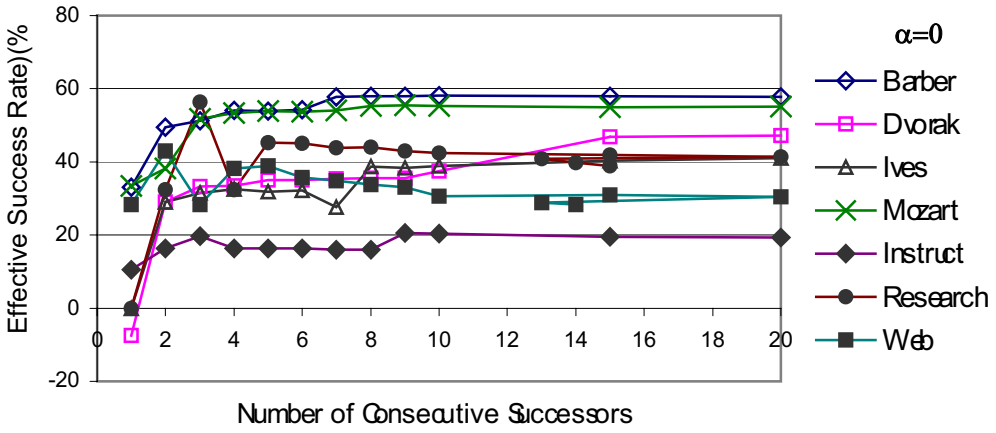


Figure 2. Effective success rate per reference of the FSS predictor for $\alpha = 0$ and *m* varying between 1 and 20.

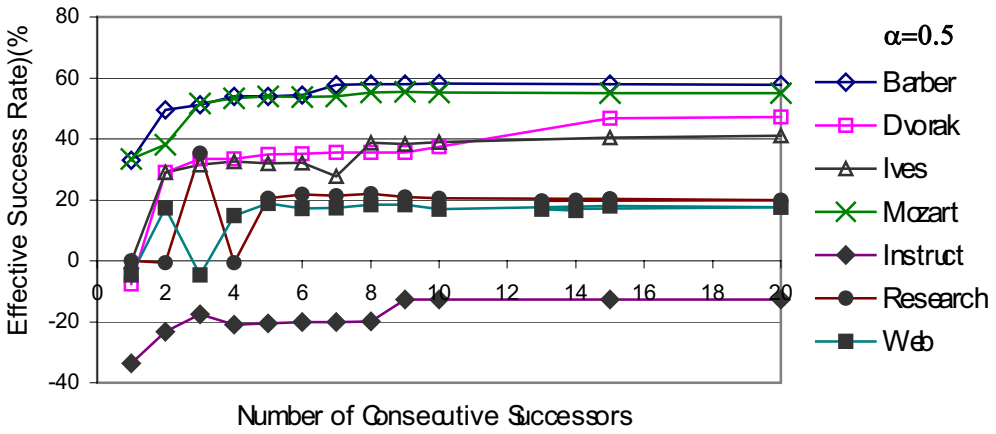


Figure 3. Effective success rate per reference of the FSS predictor for $\alpha = 0.5$ and *m* varying between 1 and 20.

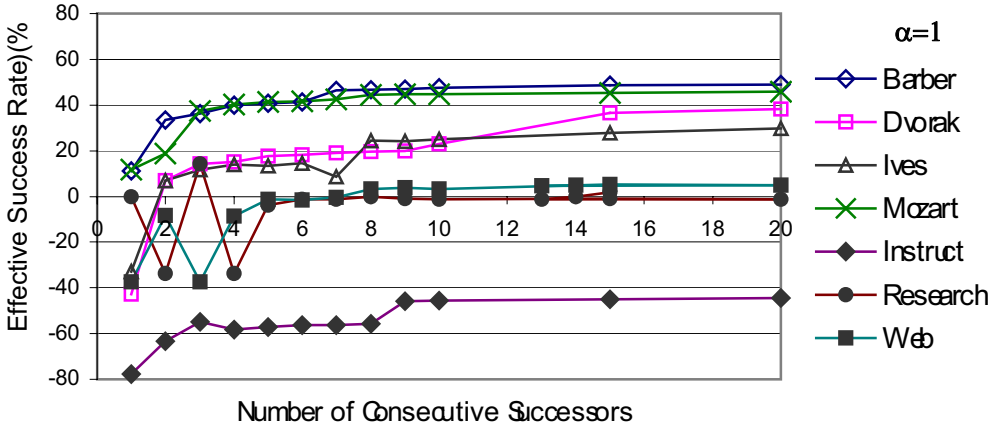


Figure 4. Effective success rate per reference of the FSS predictor for $\alpha = 1$ and *m* varying between 1 and 20.

and *barber*, a server with the highest number of system calls per second. They include between four and five million file accesses collected over a time span of approximately one year. Our second set of traces was collected in 1997 by Roselli [9] at the University of California, Berkeley over a period of approximately three months. To eliminate any interleaving issues, these traces were processed to extract the workloads of an instructional machine (*instruct*), a research machine (*research*) and a web server (*web*).

Figures 2 to 4 represent the effective success rates per reference achieved by our First Stable Successor when the number *m* of consecutive successors triggering the predictor varies between 1 and 20. Negative success rates correspond to situations where $\alpha > 0$ and the sum of the penalties assessed for incorrect predictions exceeds the number of correct predictions.

As we can see, our First Stable Successor performs much better with the four CMU traces than with the three Berkeley traces even though the Berkeley traces were collected over a much shorter period. In particular, our predictor performs very poorly with the *instruct* trace, which appears to have the least stable reference patterns of all seven traces.

The four CMU traces can be further subdivided into two groups. The first group comprises *barber* and *mozart*, which exhibit rather stable behaviors. As a result, our predictor can successfully predict between 66 and 69 percent of future references. Conversely, *dvorak* and *ives* exhibit less stable behaviors and our predictor can successfully predict between 53 and 57 percent of future references. This should not surprise us because *ives* had the largest number of users and *dvorak* the largest proportion of write activity. Even when we do not penalize incorrect predictions, First Stable Successor requires less consecutive successors to reach their optimum performance on *barber* and *mozart* than on *dvorak* and *ives*.

We can also observe that the number of consecutive successors required to achieve optimum performance increases on all seven traces when α increases from zero to one. It might be therefore indicated to increase the value of the *m* parameter for workloads that exhibit less stable file access patterns in order to reduce the number of misses.

Figures 5 to 7 compare the effective success rates per reference achieved by our First Stable Successor with $m = 8$ with those achieved by First Successor, Last Successor, Stable Successor with $m = 2$, and *k*-out-of-*m*. As we can see, our First Stable Successor predictor performs much better than First Successor but not as well as Last Successor, Stable Successor and *k*-out-of-*m*. This gap is especially evident for the *instruct* trace as these last three predictors perform almost as well as with the *mozart* trace while First Successor and First Stable Successor perform very poorly.

We can draw two major conclusions from our measurements. First, there are enough stable access patterns in the six of the seven traces we analyzed to make implicit file prefetching a worthwhile proposition. This is especially true because of the low overhead of the approach, which means that wrong predictions would only incur a minimal penalty ($\alpha \ll 1$). Second, many, if not most, of these stable access patterns are long lived and appear to persist over at least a full year. A file system implementing implicit file prefetching would probably reevaluate its file groups once a week. We can already predict that these weekly group reevaluations will not result in a complete reconfiguration of the whole file system.

5. Conclusions

Identifying and exploiting stable file access patterns is essential to the success of implicit file prefetching as this technique builds long-lived clusters of related files that can be brought into memory in a single I/O operation.

We have presented a new file access predictor that was specifically tailored to identify such stable file access patterns. Trace-driven simulation results indicate that our First Stable Successor can predict up to 70 percent of next file accesses over a period of one year.

References

[1] A. Amer and D. D. E. Long, Noah: Low-cost file access prediction through pairs, in *Proc. 20th Int' l Performance, Computing, and Communications Conf.,* pp. 27–33, Apr. 2001.

[2] A. Amer, D. D. E. Long, J.-F. Pâris, and R. C. Burns, File access prediction with adjustable accuracy, in *Proc. 21st Int' l Performance of Computers and Communication Conf.*, pp. 131–140, Apr. 2002.

[3] A. Amer, D. Long, and R. Burns. Group-based management of distributed file caches, in *Proc. 17th Int' l Conf. on Distributed Computing Systems,* pp. 525–534, July 2002.

[4] J. Griffioen and R. Appleton, Reducing file system latency using a predictive approach, in *Proc. 1994 Summer USENIX Conf.*, pp. 197–207, June 1994.

[5] T. M. Kroeger and D. D. E. Long, Design and implementation of a predictive file prefetching algorithm, in *Proc. 2001 USENIX Annual Technical Conf.*, pp. 105–118, June 2001.

[6] H. Lei and D. Duchamp, An analytical approach to file prefetching, in *Proc. 1997 USENIX Annual Technical Conf.*, pp. 305-318, Jan. 1997.

[7] L. Mummert and M. Satyanarayanan, Long term distributed file reference tracing: implementation and experience, Technical Report, School of Computer Science, Carnegie Mellon University, 1994.

[8] M. L. Palmer and S. B. Zdonik, FIDO: a cache that learns to fetch, in *Proc. 17th Int' l Conf. on Very Large Data Bases,* pp. 255–264, Sept. 1991.

[9] D. Roselli, Characteristics of file system workloads, Technical. Report CSD-98-1029, University of California, Berkeley, 1998.

[10] E. Shriver, C. Small, and K. A. Smith, Why does file system prefetching work? in *Proc. 1999 USENIX Technical Conf.*, pp. 71–83**,** June 1999.

[11] C. Tait and D. Duchamp, Detection and exploitation of file working sets, in *Proc. 11th Int' l Conf. on Distributed Computing Systems*, pp. 2–9, May 1991.

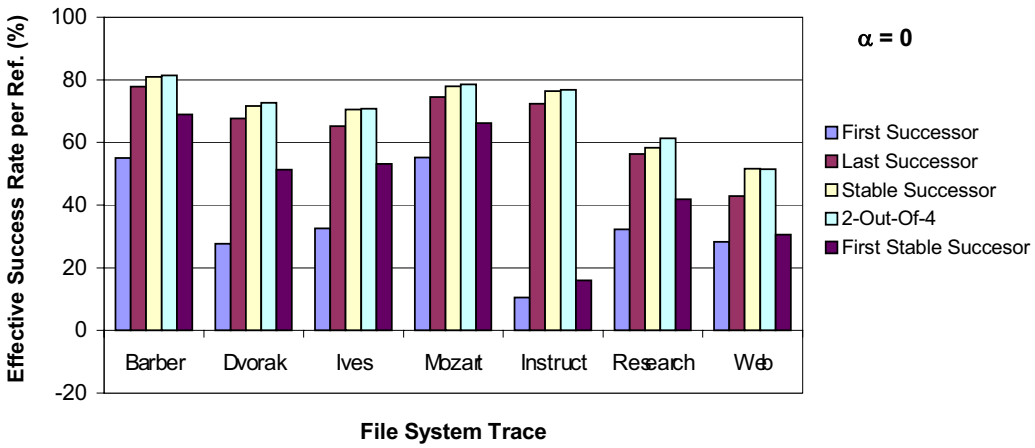


Figure 5. Compared success rates per reference of the five policies for $\alpha = 0$.

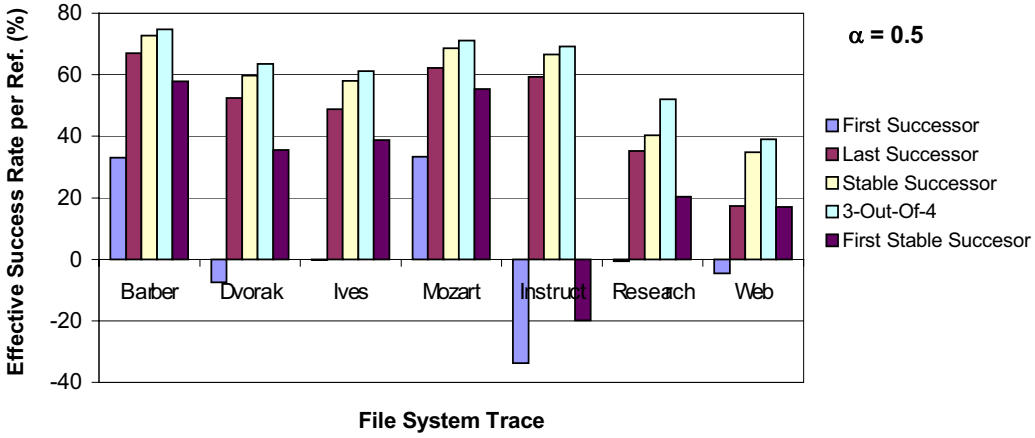


Figure 6. Compared success rates per reference of the five policies for $\alpha = 0.5$.

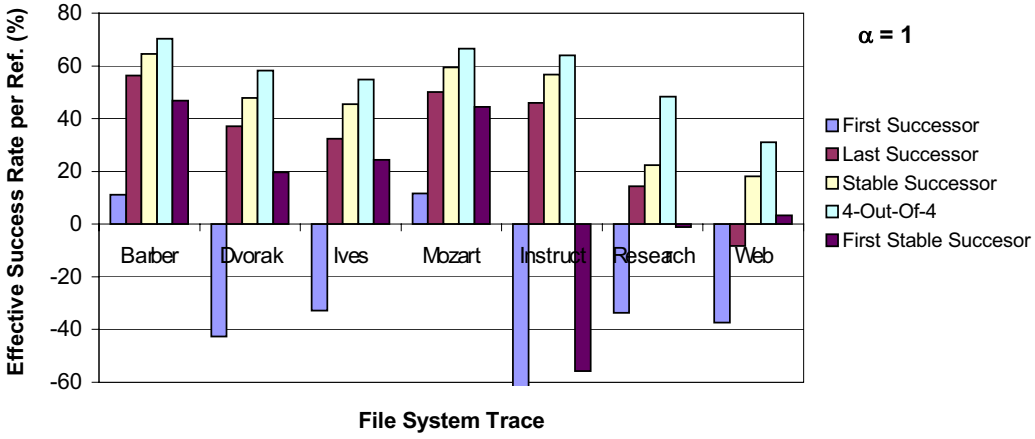


Figure 7. Compared success rates per reference of the five policies for $\alpha = 1$.

An On-line Backup Function for a Clustered NAS System (X-NAS)

Yoshiko Yasuda, Shinichi Kawamoto, Atsushi Ebata, Jun Okitsu, and Tatsuo Higuchi
Hitachi, Ltd., Central Research Laboratory
1-280 Higashi-koigakubo, Kokubunji-shi, Tokyo 185-8601, Japan
Tel: +81-423-23-1111, Fax: +81-423-27-7743
e-mail: {yoshikoy, skawamo, ebata, j-okitsu, higuchi}@crl.hitachi.co.ip

Abstract
An on-line backup function for X-NAS, a clustered NAS system designed for entry-level NAS, has been developed. The on-line backup function can replicate file objects on X-NAS to a remote NAS in real-time. It makes use of the virtualized global file system of X-NAS, and sends NFS write operations to both X-NAS and the remote backup NAS at the same time. The performance of the on-line backup function was evaluated and the evaluation results show that the on-line backup function of X-NAS improves the system reliability while maintaining 80% of the throughput of the X-NAS without this function.

1. Introduction

An entry-level NAS system is convenient in terms of the cost and the ease of management for offices with no IT experts. However, it is not scalable. To solve this problem, X-NAS, which is a simple, scalable clustered NAS architecture designed for entry-level NAS, has been proposed [6]. Like conventional NAS systems, it can be used for various clients, such as those using UNIX and Windows[1]. X-NAS aims at the following four goals.

- Cost reduction by using entry-level NAS as an element
- Ease of use by providing a single-file-system view for various kinds of clients
- Ease of management by providing a centralized management function
- Ease of scaling-up by providing several system-reconfiguration functions

To achieve these goals, X-NAS virtualizes multiple entry-level NAS systems as a unified system without changing clients' environments. In addition, X-NAS maintains the manageability and the performance of the entry-level NAS. It also can easily be reconfigured without stopping file services or changing setting information. However, when one of the X-NAS elements suffers a fault, file objects on the faulty NAS system may be lost if there are no backups. To improve the X-NAS reliability, a file-replication function must therefore be developed.

The goal of the present work is to introduce an on-line backup function of X-NAS that replicates original file objects on X-NAS to a remote NAS for each file access request in real-time without changing the clients' environments. The performance of the on-line backup function was evaluated and the evaluation results indicate that X-NAS with the on-line backup function improves the system reliability while maintaining 80% of the throughput of standard X-NAS.

[1] Windows and DFS are trademarks of Microsoft Corporation. Double Take is a trademark of Network Specialists, Inc. All other products are trademarks of their respective corporations.

2. On-line backup function for X-NAS

To improve the reliability of X-NAS, an on-line backup function for X-NAS has been developed. (Since the details of the X-NAS structure are discussed in another paper [6], they are not described here.) The on-line backup function consists of many sub-functions. Among these sub-functions, we focus on on-line replication, the heart of the on-line backup function, in this paper. The on-line replication replicates files of X-NAS to a remote NAS, which is called a backup NAS, in real-time for each file access request.

2.1. Requirements

The on-line backup function of X-NAS must meet the following requirements:

- Generate replicas of file objects in real-time in order to eliminate the time lag between the original data and the replicas.
- Use a standard file-access protocol such as NFS to communicate between X-NAS and the backup NAS in order to apply as many kinds of NAS as clients need.
- Do not change clients' environments in order to curb their management cost.

2.2. On-line replication

There are several methods for replicating file objects to remote systems via an IP network. One method is to use a block I/O [5]. Since using a block I/O is a fine-grain process, all file objects are completely consistent with copied objects. However, the system structure is limited because the logical disk blocks of the objects must be allocated to the same address between the original data and its replica. Another method is to change the client's system. DFS [1] is a simple method for replicating file objects to many NASs. It replicates file objects in constant intervals but not in real-time.

Xnfsd and the management partition in X-NAS enable the centralized management of many NAS elements and provide a unified file system view for clients (Fig. 1). Xnfsd is a wrapper daemon and receives an NFS operation in place of the NFS server and sends the operation to others. On-line replication of X-NAS makes use of Xnfsd in order to copy file objects to the backup NAS. By extending this function, Xnfsd sends the NFS operation not only to the NFS servers on the X-NAS but also to the NFS servers on the backup NAS. All file objects can thus be replicated in real-time for each NFS operation.

2.2.1. Operations

NFS operations handled in X-NAS can be divided into four categories. Category 1 is reading files; category 2 is writing files; category 3 is reading directories; and category 4 is writing directories. Xnfsd sends NFS operations belonging to categories 2 and 4 to both X-NAS and the backup NAS at the same time. On the other hand, NFS operations belonging to categories 1 and 3 are not sent to the backup NAS.

When a UNIX client sends a WRITE operation for file *f* to X-NAS, Xnfsd on P-NAS (parent NAS) receives the operation in place of the NFS daemon. Figure 1 shows the flow of this operation, and Figure 2 shows the timing chart with or without the on-line backup function. Firstly, Xnfsd specifies a data partition that stores the file entity by using the inode number of the dummy file *f* on the management partition (#1). Secondly, Xnfsd invokes a sub thread and then sends the WRITE operation to the backup NAS by

using the thread (#2). Thirdly, Xnfsd sends the WRITE operation to the NFS daemon on the specified C-NAS (child NAS), and then the C-NAS processes the operation (#3). Finally, Xnfsd waits for the responses of the operations from the NFS server on the C-NAS and from the backup NAS (#4), and then it makes one response from all the responses and sends it back to the client. We call this procedure a synchronized backup.

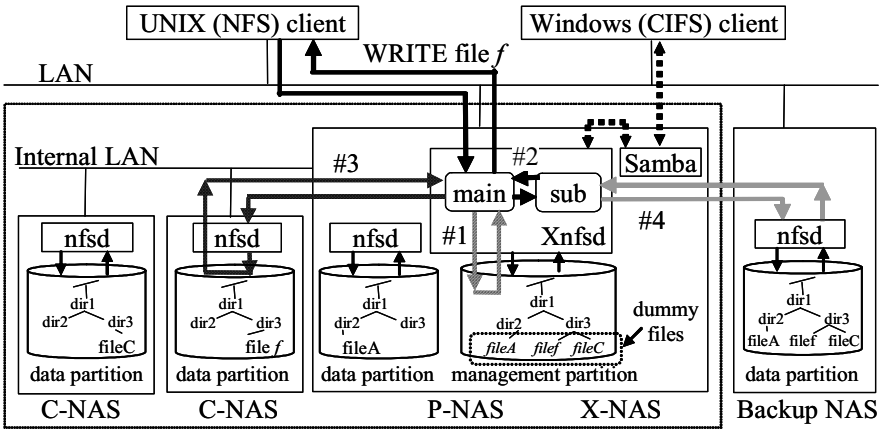


Figure 1: Flow of WRITE operation with online backup function.

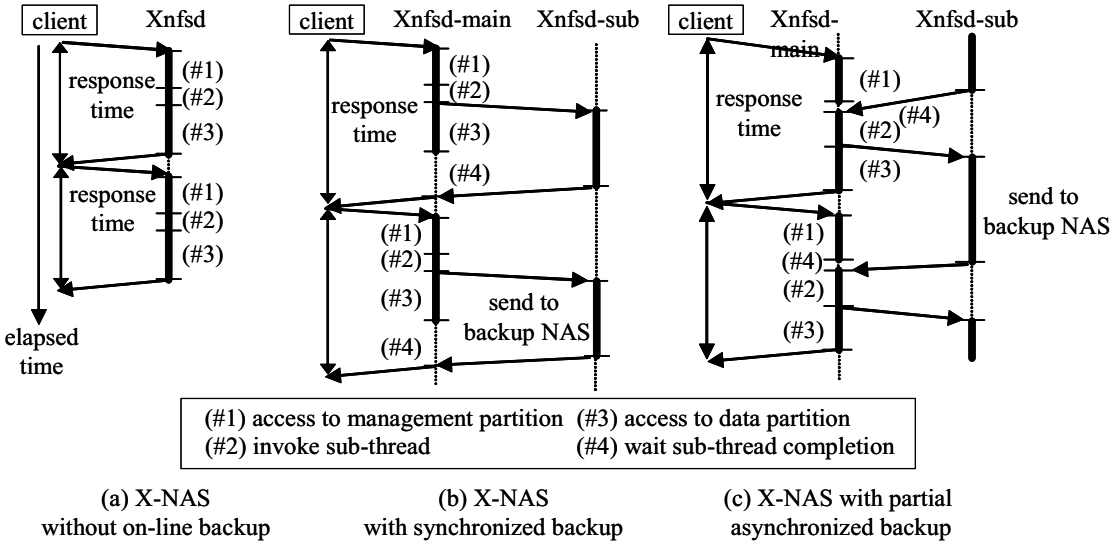


Figure 2. Timing charts of WRITE operation with or without on-line backup function.

2.2.2. Key features

An on-line backup function must guarantee the consistency of data between X-NAS and the backup NAS. To achieve this, Xnfsd waits for all responses from both one of the NFS servers on the X-NAS and the backup NAS for each NFS operation. However, waiting for the responses degrades total performance. To solve this problem, the performance of the on-line replication function must be improved through three key features as follows.

(1) Multi-threaded wrapper daemon

Xnfsd waits for all responses from both NAS systems. This incurs an overhead because of frequent accesses to the network and the disk drives. To reduce this cost, the main

thread of Xnfsd invokes a sub thread to send the file I/Os to the backup NAS. This feature enables X-NAS to process the disk accesses of both X-NAS and the backup NAS in parallel.

(2) File-handle cache
The cost of specifying the full path name and the file handle on the backup NAS is high because of frequent accesses to the network and the disk drives. To reduce this cost, X-NAS makes use of a file-handle cache, which records the correspondence between the file handle of the dummy file, i.e., the global file handle, and the file handle of the backup NAS.

(3) Partial asynchronized backup
Although the synchronized backup is a simple method, the execution cost is high because this method waits for all the responses from the NFS servers on the X-NAS and the backup NAS. A method that does not wait for the response from the backup NAS achieves the same performance as X-NAS without the on-line backup function. However, when X-NAS or the backup NAS becomes faulty, it is difficult to guarantee the consistency of data between X-NAS and the backup NAS. Using a log is one solution to guarantee the consistency. However, since the log size is limited, it is not a perfect solution for entry-level NAS, which usually has a small-sized memory. Furthermore, according to the X-NAS concept, the architecture must be simplified as much as possible. Xnfsd thus supports a partial asynchronized backup method in addition to the synchronized backup. Figure 2(c) shows the timing chart of the WRITE operation with partial asynchronized backup. In the method, after processing disk accesses to the data partition on the X-NAS element, Xnfsd sends back a response to a client without waiting for the response from the backup NAS. As a result, the client can send the next operation. The main thread of Xnfsd can perform the disk accesses to the management partition for the next operation during the waiting time for the response from the backup NAS.

3. Performance evaluation

To evaluate the on-line backup function of X-NAS, an X-NAS prototype based on the NFSv3 implementation was developed. We ran NetBench [3] and SPECsfs97 [4] on the X-NAS prototype with or without on-line backup function. In this evaluation, by taking account of permissible range for the entry-level NAS's users, we set the performance objective for X-NAS with the on-line backup function at 80% of the performance of X-NAS without the function. Throughput and average response time are used as the performance metrics. In this evaluation, we implemented the partial asynchronized backup function in the WRITE operation. This is because the ratio of the WRITE operations to all operations is higher than other operations in the workload mix of the benchmarks. Furthermore, since the file sizes used by the benchmark programs are from 100 to 300 KB, many WRITE operations are issued continuously and then each process in a WRITE operation could be overlapped.

3.1. Experimental environment

In the experimental environment, the maximum number of X-NAS elements is fixed to four. Each X-NAS element and the backup NAS configured with one 1-GHz Pentium III

processor, 1 GB of RAM and a 35-GB Ultra 160 SCSI disk drive running Red Hat Linux 7.2. For the NetBench test, one to eight clients running Windows 2000 Professional were used. The clients, P-NAS, C-NASs, and the backup NAS were connected by 100-Megabit Ethernet because most offices still use this type of LAN.

3.2. Results

Figures 3 and 4 show the results of our performance evaluation in terms of throughput and average response time. The throughputs of X-NAS with the synchronized backup function are about 80% of those without the function. Under an experimental environment with NetBench, the average response time for X-NAS with the function is about 1.2 times higher than that for X-NAS without the function. On the other hand, under an experimental environment with SPECsfs, the average response time for X-NAS with the function is about 1.4 times higher than the time for X-NAS without it. Although the partial asynchronized backup can improve both throughput and average response time by several percentage, the performance objective for the response time in the case of SPECsfs cannot be achieved yet.

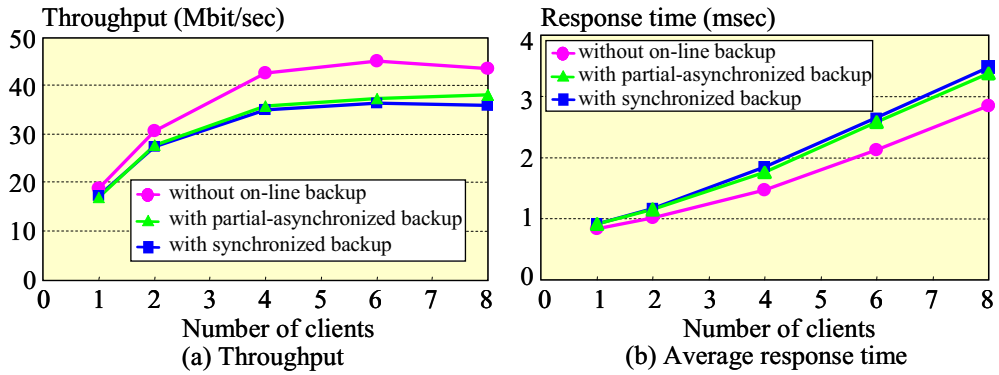


Figure 3. Throughput and average response time of X-NAS with or without on-line backup function in the case of NetBench.

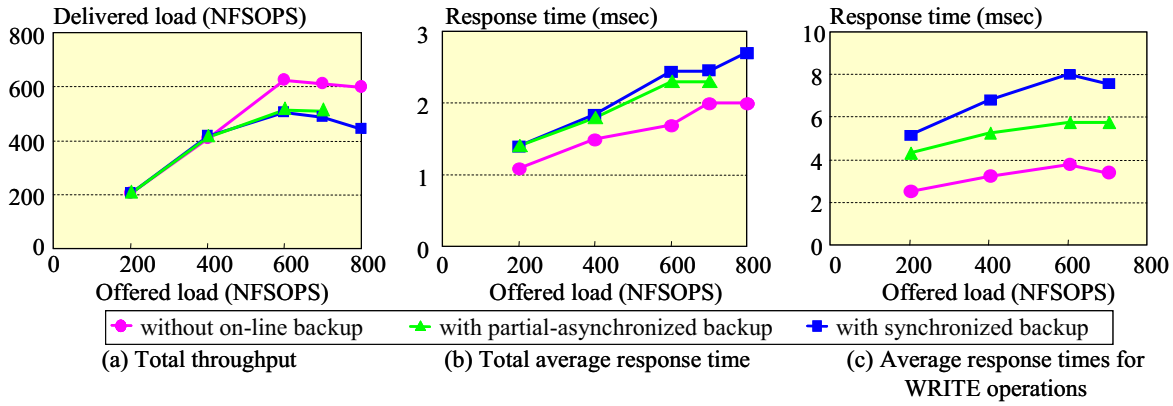


Figure 4. Throughput and average response time of X-NAS with or without on-line backup function in the case of SPECsfs.

3.3. Discussion

To specify the reason for the longer response time in the case of SPECsfs, the average

response time of each NFS operation in the case of the synchronized backup was analyzed. The average response times for some write requests such as WRITE, SETATTR and CREATE are longer than those for X-NAS without that function. In particular, the average response time for WRITE operations is 2.5 times higher than that for the other operations. Profiling results of the WRITE operations shows that waiting time for the sub-thread completion is about 24% of the total processing time and access to the data partition via an IP network is about 48% of that time. By applying the partial asynchronized backup to X-NAS, this waiting time can be reduced to almost zero. Figure 4(c) shows the effects of the partial asynchronized backup in the case of WRITE operations. The average response time for WRITE operations with the partial asynchronized backup can be reduced to from 2.5 times to 1.8 times the time for X-NAS without the function. As a result, the total average response time for SPECsfs with the function can be reduced to 1.3 times that without it. However since the ratio of data transmission time to the total processing time is still higher in the case of the 100-Megabit Ethernet, using a Gigabit network is effective because it can reduce the data transmission time for 100-Megabit Ethernet to at least one-fifth. Furthermore, by optimizing other operations such as CREATE and COMMIT, the performance objective of 1.2 times can be achieved.

4. Related work

There are several methods for replicating file objects between several NAS systems via the network. DFS [1] is a simple and easy file-replication function on Windows systems. DRBD [5] is a kernel module for building a two-node HA cluster under Linux. Double Take [2] is a third-vendor software to replicate file objects on the master NAS to the slave NAS.

5. Conclusions

An on-line backup function for X-NAS, a clustered NAS system, has been developed. On-line replication, the core of the on-line backup function, replicates file objects on X-NAS to a remote backup NAS in real-time for each NFS operation. A multi-threaded wrapper daemon with a low overhead, the developed file-handle cache and the partial asynchronized backup method can reduce the overhead for accessing the backup NAS. An X-NAS prototype with the on-line backup function, based on NFSv3 running the NetBench and SPECsfs97 programs attains 80% of the performance of X-NAS without the function. This function improves the dependability of entry-level NAS while maintaining its manageability.

References

[1] Deploying Windows Powered NAS Using Dfs with or without Active Directory. http://www.microsoft.com, 2001.

[2] Double-Take Theory of Operations. http://www.nsisoftware.com, 2001.

[3] NetBench 7.0.3. http://www.etestomglabs.com/benchmarks/netbench, 2002.

[4] SFS3.0 Documentation Version 1.0. http://www.spec.org, 2002.

[5] P. Reisner. DRBD. In Proceedings of the 7th International Linux Kongress, 2000.

[6] Y. Yasuda et al. Concept and Evaluation of X-NAS: a highly scalable NAS system. In Proceedings of the 20th IEEE/11th NASA MSST2003.

dCache, the commodity cache

Patrick Fuhrmann
Deutsches Elektronen Synchrotron
22607 Hamburg, Germany
Notkestrasse 85
Tel: +49-40-8998-4474, Fax: +49-40-8994-4474
e-mail: patrick.fuhrmann@desy.de

1. Abstract
The software package presented within this paper has proven to be capable of managing the storage and exchange of several hundreds of terabytes of data, transparently distributed among dozens of disk storage nodes. One of the key design features of the dCache is that although the location and multiplicity of the data is autonomously determined by the system, based on configuration, cpu load and disk space, the name space is uniquely represented within a single file system tree. The system has shown to significantly improve the efficiency of connected tape storage systems, through caching, 'gather & flush' and scheduled staging techniques. Furthermore, it optimizes the throughput to and from data clients as well as smoothing the load of the connected disk storage nodes by dynamically replicating datasets on the detection of load hot spots. The system is tolerant against failures of its data servers which enables administrators to go for commodity disk storage components. Access to the data is provided by various ftp dialects, including gridftp, as well as by a native protocol, offering regular file system operations like open/read/write/seek/stat/close. Furthermore the software is coming with an implementation of the Storage Resource Manager protocol, SRM, which is evolving to an open standard for grid middleware to communicate with site specific storage fabrics.

2. Contributors
The software is being developed by the Deutsches Elektronen Synchrotron (DESY) in Hamburg, Germany[1] and the Fermi National Laboratory, Batavia Chicago,IL, USA [2].

3. Technical Specification

3.1 File name space and dataset location
dCache strictly separates the filename space of its data repository from the actual physical location of the datasets. The filename space is internally managed by a database and interfaced to the user resp. to the application process by the nfs2 [9] protocol and through the various ftp filename operations. The location of a particular file may be on one or more dCache data servers as well as within the repository of an external Tertiary Storage Manager. dCache transparently handles all necessary data transfers between nodes and optionally between the external Storage Manager and the cache itself. Inter dCache transfers may be caused by configuration or load balancing constrains. As long as

a file is transient, all dCache client operations to the dataset are suspended and resumed as soon as the file is fully available.

3.2Maintenance and fault tolerance

As a result of the name space —data separation, dCache data server nodes subsequently denoted as pools, can be added at any time without interfering with system operation. Having a Tertiary Storage System attached, or having the system configured to hold multiple copies of each dataset, data nodes can even be shut down at any time. Under those conditions, the cache system is extremely tolerant against failures of its data server nodes.

3.3Data access methods

In order to access dataset contents, dCache provides a native protocol (dCap), supporting regular file access functionality. The software package includes a c-language client implementation of this protocol offering the posix open/read/write/seek/stat/close calls. This library may be linked against the client application or may be preloaded to overwrite the file system I/O operations. The library supports pluggable security mechanisms where the GssApi (Kerberos) and ssl security protocols are already implemented. Additionally, it performs all necessary actions to survive a network or pool node failure. It is available for Solaris, Linux, Irix64 and windows. Furthermore, it allows to open files using an http like syntax without having the dCache nfs file system mounted. In addition to this native access, various FTP dialects are supported, e.g. GssFtp (kerberos) [8] and GsiFtp (GridFtp) [7]. An interface definition is provided, allowing other protocols to be implemented as well.

3.4Tertiary Storage Manager connection

Although dCache may be operated stand alone, it can also be connected to one or more Tertiary Storage Systems. In order to interact with such a system, a dCache external procedure must be provided to store data into and retrieve data from the corresponding store. A single dCache instance may talk to as many storage systems as required. The cache provides standard methods to optimize access to those systems.

Whenever a dataset is requested and cannot be found on one of the dCache pools, the cache sends a request to the connected Tape Storage Systems and retrieves the file from there. If done so, the file is made available to the requesting client. To select a pool for staging a file, the cache considers configuration information as well as pool load, available space and a Least Recently Used algorithms to free space for the incoming data.

Data, written into the cache by clients, is collected and, depending on configuration, flushed into the connected tape system based on a timer or on the maximum number of bytes stored, or both. The incoming data is sorted, so that only data is flushed which will go to the same tape or tape set.

Mechanisms are provided that allow giving hints to the cache system about which file will be needed in the near future. The cache will do its best to stage the particular file before it’s requested for transfer.

Space management is internally handled by the dCache itself. Files which have their origin on a connected tape storage system will be removed from cache, based on a Least

Recently Used algorithm, if space is running short. Space is created only when needed. No high/low watermarks are used.

3.5 Pool Attraction Model

Though dCache distributes datasets autonomously among its data nodes, preferences may be configured. As input, those rules can take the data flow direction, the subdirectory location within the dCache file system, storage information of the connected Storage Systems as well as the IP number of the requesting client. The cache defines data flow direction as getting the file from a client, delivering a file to a client and fetching a file from the Tertiary Storage System. The simplest setup would direct incoming data to data pools with highly reliable disk systems, collect it and flush it to the Tape Storage System when needed. Those pools could e.g. not be allowed to retrieve data from the Tertiary Storage System as well as deliver data to the clients. The commodity pools on the other hand would only handle data fetched from the Storage System and delivered to the clients because they would never hold the original copy and therefore a disk /node failure wouldn't do any harm to the cache. Extended setups may include the network topology to select an appropriate pool node. Those rules result in a matrix of pools from which the load balancing module, described below, may choose the most appropriate candidate. Each row of the matrix contains pools with similar attraction. Attraction decreases from top to bottom. Should none of the pools in the top row be available, the next row is chosen, a.s.o.. Optionally, stepping from top to bottom can be done as long as the candidate of row 'n' is still above a certain load. The final decision, which pool to select of this set, is based on free space, age of file and node load considerations.

3.6 Load Balancing and pool to pool transfers

The load balancing module is, as described above, the second step in the pool selection process. This module keeps itself updated on the number of active data transfers and the age of the least recently used file for each pool. Based on this set of information, the most appropriate pool is chosen. This mechanism is efficient even if requests are arriving in bunches. In other words, as a new request comes in, the scheduler already knows about the overall state change of the whole system triggered by the previous request though this state change might not even have fully evolved. System administrators may decide to make pools with unused files more attractive than pools with only a small number of movers, or some combination. Starting at a certain load, pools can be configured to transfer datasets to other, less loaded pools, to smooth the overall load pattern. At a certain point, pools may even fetch a file from the Tertiary Storage System again, if all pools, holding the requested dataset are too busy. Regulations are in place to suppress chaotic pool to pool transfer orgies in case the global load is steadily increasing. Furthermore, the maximum numbers of replica of the same file can be defined to avoid having the same set of files on each node.

3.7 File Replica Manager

A first version of the so called Replica Manager is currently under evaluation. This module enforces that at least N copies of each file, distributed over different pool nodes, must exist within the system, but never more than M copies. This approach allows to shut

down servers without affecting system availability or to overcome node or disk failures. The administration interface allows to announce a scheduled node shut down to the Replica Manager so that it can adjust the N — M interval.

4. Data Grid functionality

In the context of the LHC Computing Grid Project [4], a Storage Element describes a module providing mass data to local Computing Elements. To let a local Storage System look like a Storage Element, two conditions must be met. Storage Elements must be able to communicate to each other in order to exchange mass data between sites running different Storage System and Storage Elements have to provide local data through standard methods to allow GRID jobs to access data files in a site independent manner. The first requirement is covered by a protocol called the Storage Resource Manager, SRM [3], defining a set of commands to be implemented by the local Storage System to enable remote access. It mainly covers queries about the availability of datasets as well as commands to prepare data for remote transfer and to negotiate appropriate transfer mechanisms. dCache is providing an SRM interface and has proven to be able to talk to other implementations of the SRM. A dCache system at FERMI is successfully exchanging data with the CASTOR Storage Manager at CERN using the SRM protocol for high level communication and GridFtp for the actual data transfer. The second requirement, to make local files available to Grid Applications, is approached by the G-File initiative, a quasi standard as well. It offers well defined, posix like function calls that allow site independent access to files held by the local Storage Element. Optionally G-File can talk to other grid modules to register imported files or files being exportable. G-File developers at CERN have successfully linked the g-file library against the dCache dCap library.

5. Dissemination

In the meantime, dCache is in production at various locations in Europe and the US. The largest installation is, to our knowledge, the CDF system at FERMI [2]. 150 Tbytes are stored on commodity disk systems and in the order of 25 Tbytes have been delivered to about 1000 clients daily for more than a year. FERMI dCache installations are typically connected to ENSTORE [11], the FERMI tape storage system. CDF is operating more than 10 tape-less dCache installations outside of FERMI, evaluating the dCache Replica Manager. The US devision of the LHC CMS [13] experiment is using the dCache as Grid Storage Element and large file store in the US and Europe. At DESY, dCache is connected to the Open Storage Manager (OSM) and serving data out of 70 Tbytes of disk space. The German LHC Grid Tier 1 center in Karlruhe (GridKa,[12]) is in the process of building a dCache installation as Grid Storage Element, connected to their Tivoli Storage Manager [16] installation.

6. References

[1] DESY : http://www.desy.de
[2] FERMI : http://www.fnal.gov
[3] SRM : http://sdm.lbl.gov/srm-wg
[4] LCG : http://lcg.web.cern.ch/LCG/

[5] CASTOR Storage Manager : http://castor.web.cern.ch/castor/
[6] dCache Documentation : http://www.dcache.org
[7] GsiFtp : http://www.globus.org/datagrid/deliverables/gsiftp-tools.html
[8] Secure Ftp : http://www.ietf.org/rfc/rfc2228.txt
[9] NFS2 : http://www.ietf.org/rfc/rfc1094.txt
[10] Fermi CDF Experiment : http://www-cdf.fnal.gov
[11] Fermi Enstore : http://www.fnal.gov/docs/products/enstore/
[12] GridKA : http://www.gridka.de/
[13] Cern CMS Experiment : http://cmsinfo.cern.ch
[14] Cern LHC Project : http://lhc.web.cern.ch/LHC
[15] Grid g-file :
http://lcg.web.cern.ch/LCG/peb/GTA/GTA-ES/Grid-File-AccessDesign-v1.0.doc
[16] Tivoli Storage Manager :
http://www-306.ibm.com/software/tivoli/products/storage-mgr/

HIERARCHICAL STORAGE MANAGEMENT AT THE NASA CENTER FOR COMPUTATIONAL SCIENCES: FROM UNITREE TO SAM-QFS

Ellen Salmon, Adina Tarshish, Nancy Palm
NASA Center for Computational Sciences (NCCS)
NASA Goddard Space Flight Center (GSFC), Code 931
Greenbelt, Maryland 20071
Tel: +1-301-286-7705
e-mail: Ellen.M.Salmon@nasa.gov

Sanjay Patel, Marty Saletta, Ed Vanderlan, Mike Rouch, Lisa Burns, Dr. Daniel Duffy
Computer Sciences Corporation, NCCS GSFC
Greenbelt, Maryland 20071
Tel: +1-301-286-3131
e-mail: sjpatel@calvin.gsfc.nasa.gov

Robert Caine, Randall Golay
Sun Microsystems, Inc.
7900 Westpark Drive
McLean, VA, 22102
Tel: +1-703-280-3952
e-mail: Robert.Caine@sun.com

Jeff Paffel, Nathan Schumann
Instrumental, Inc.
2748 East 82nd Street
Bloomington, MN 55425
Tel: +1-715-832-1499
e-mail: jpaffel@instrumental.com

Abstract
This paper presents the data management issues associated with a large center like the NCCS and how these issues are addressed. More specifically, the focus of this paper is on the recent transition from a legacy UniTree (Legato) system to a SAM-QFS (Sun) system. Therefore, this paper will describe the motivations, from both a hardware and software perspective, for migrating from one system to another. Coupled with the migration from UniTree into SAM-QFS, the complete mass storage environment was upgraded to provide high availability, redundancy, and enhanced performance. This paper will describe the resulting solution and lessons learned throughout the migration process.

1. Introduction

The Science Computing Branch of the Earth and Space Data Computing Division at the Goddard Space Flight Center (GSFC) manages and operates the NASA Center for

Computational Sciences (NCCS).[1] The NCCS is a shared center providing supercomputing services and petabyte-capacity data storage to a variety of user groups. Its mission is to enable Earth and space sciences research through computational modeling by providing its user community access to state of the art facilities in High Performance Computing (HPC), mass storage technologies, high-speed networking, and HPC computational science expertise.

The largest workloads currently being performed at the NCCS consist of Earth system and climate modeling, prediction, and data assimilation. Input data for these applications come from many sources, including ground and satellite stations. Both computer and sensor technology have grown dramatically within the last decade causing a boom in the amount of data generated by these types of sources.[2]

The major groups that comprise the NCCS user community include the following:

- *Global Modeling and Assimilation Office (GMAO)*: consists of both the Seasonal-to-Interannual Prediction Project (NSIPP) and the Data Assimilation Office (DAO), produces ensembles of simulations of near-term climate and creates research-quality assimilated global data sets from multiple satellites for climate analysis and observation planning.
- *Goddard Institute for Space Studies (GISS)*: produces climate studies focusing on timescales ranging from a decade to a century.
- *ESTO/Computational Technologies Project*: develops the Earth System Modeling Framework (ESMF).
- *Atmospheric Chemistry*: research teams investigating the evolution of the composition of the Earth's atmosphere and its impact on weather and climate.
- *Research and Analysis Group*: a large collection of smaller research efforts.

2. Data Management at the NCCS

With over 3 Teraflops of computational capacity, the research performed throughout the heterogeneous environment of the NCCS uses large amounts of existing data for new computational studies while generating large amounts of new data from the output of these studies. In general, the total data stored at the NCCS is growing at approximately 125 TB of data per year, which includes both primary and secondary copies of user data. As an example of this net growth of data, during FY03, a total of 207 TB of new data was stored while approximately 143 TB of data was deleted. This resulted in a net growth of 64 TB of single copy data or 128 TB when duplicated. Complementary, the number of files managed by the Mass Data Storage and Delivery System (MDSDS) has grown from 3.5 million in 1999 to more than 10 million in 2003.

Figure 1 shows the linear data growth as measured at the end of the fiscal year (month of September) for the past five years of only the legacy UniTree data. This trend is expected to increase dramatically in the next few years as the diverse mass storage facilities at the NCCS are consolidated and with increased utilization of the computational resources.

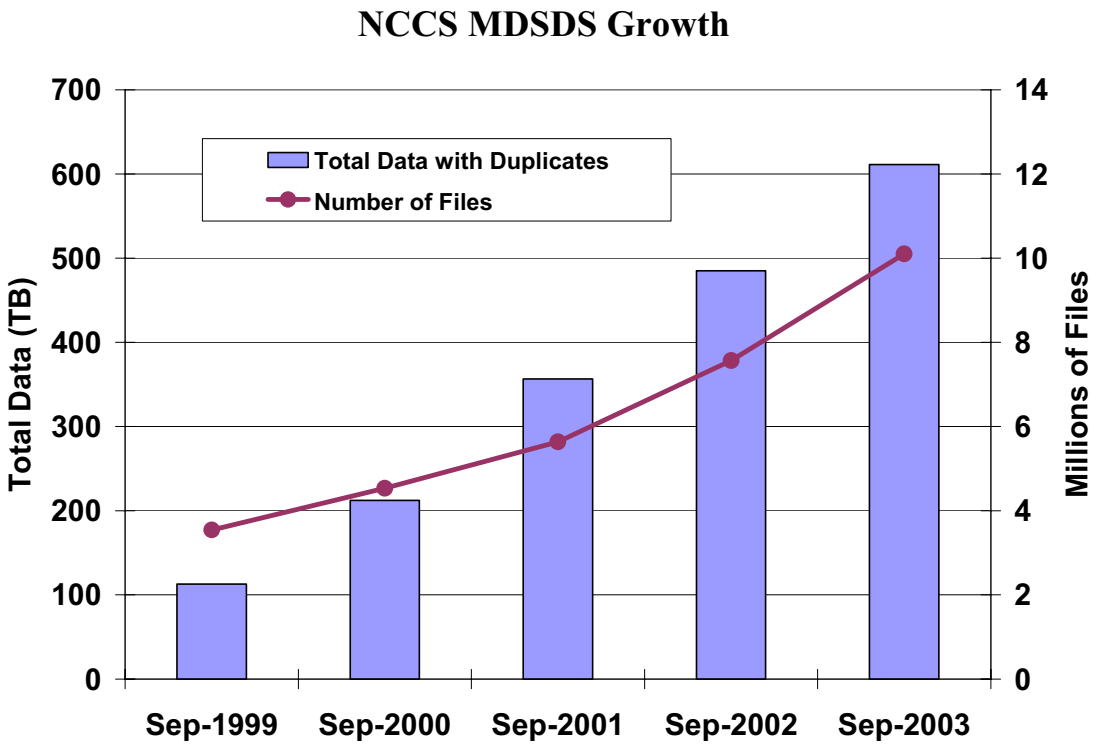


Figure 1: Growth of Mass Data Storage and Delivery System (MDSDS) data and files at the NCCS.

Throughout any given day, data is pulled from and stored to the MDSDS as jobs are run on the computational platforms. The NCCS measures the inbound and outbound data of the mass storage system and has seen traffic, files being transferred into and out of the mass storage system, of up to a total of 2.9 TB for a single day. Therefore, the resulting storage system must not only keep pace with the increase in overall storage, but also maintain the capability to serve larger amounts of data on demand.

3. Hierarchical Storage Management (HSM)

An HSM consists of different layers of storage capability for users to store and retrieve their data. Typically, a high speed disk cache is used as the first layer of storage and software is run to migrate files from high speed disk cache (1st layer storage) to slower tape media (2nd layer storage).

There are two existing HSM systems at the NCCS. The NCCS provides the MDSDS, previously running UniTree, for high-performance long-term storage for most NCCS user data.[3,4] A second system, which uses the SGI Data Migration Facility (DMF), supports the GMAO DAO users. This paper only discusses the replacement of the MDSDS's OTG's DiskXtender Storage Manager (DXSM) software, formerly known as UniTree Central File Manager (UCFM).

The UniTree management software runs on an aging Sun E10K with eight TB of Data Direct Network high performance disk storage. The MDSDS manages eight StorageTek (STK) Powderhorn 9310 robotic silos (five primary silos in the NCCS's primary building

and three secondary risk-mitigation silos in a building a mile away). For all user data, the MDSDS is configured to make a primary copy of files on tapes in the NCCS's primary building and a secondary copy on the tapes in the risk mitigation location.

Users access the MDSDS through the File Transfer Protocol (FTP) from any of the computational platforms or even their desktops. A home directory for each user is defined within a single MDSDS file system. Files that are put into UniTree are first copied into the MDSDS file system and then archived to tape and later released from disk cache according to NCCS policies. File retrieves are transparent whether the file resides on disk or must first be staged from tape; however, any retrieves of files from tapes incur a latency to load the tape into a tape drive, position the tape to the beginning of the file, and then copy the file to the disk.

While UniTree was a very reliable mass storage software system, by the middle of 2001, it became apparent that the recently modified capacity license cost model for UniTree was not compatible with the NCCS budget in light of the NCCS users' projected growth over the ensuing years. The NCCS began exploring alternatives and undertook a detailed feature comparison of four major storage management systems used for several years in high performance computing environments. The candidates were SGI's Data Migration Facility (DMF), IBM's High Performance Storage System (HPSS), Sun's SAM-QFS (also known as Sun StorEdge Performance and Utilization Suite), and UniTree. The various solutions were evaluated based on the following attributes:

- *Performance*: meet the needs for user requests for storage and retrieval of data.
- *Integrity/High Availability*: stable and safe environment more readily available than the existing HSM.
- *Flexible/Modular/Scalable*: allows for the maximum possible options for hardware and software and can scale with the users' requirements.
- *Balance*: avoid bottlenecks throughout the flow of data to the storage media.
- *Manageable*: tools provide a rich environment for administration and reporting.

4. Sun/SAM-QFS Solution

An internal panel evaluated the vendor responses and awarded the highest rating to the Sun SAM-QFS proposal. Notably, the Sun proposal scored high marks for its ability to be configured for high availability by sharing file systems in a clustered environment, its ability to "stream" the writing of tiny files to tape by combining them into "containers," and by having the largest customer base. Complementary to the Sun proposal, the NCCS also purchased more disk space and tape drive upgrades. The resulting system continued to leverage the existing investment in the STK hardware while providing a viable system to meet the future needs of the NCCS.

A Sun Fire 15K system was purchased and configured into two distinct domains. These two domains, along with multiple interfaces to each, provide a highly available system for the user community. Fully redundant, SAM-QFS provides the necessary storage management software to provide multiple file systems with storage, archive management,

and retrieval capabilities for a variety of storage media. The major components that make up the Sun SAM-QFS software are as follows:[6]

- *Archiver*: automatically copies online disk cache files to archive media. The archive media can consist of either online disks or removable media cartridges.
- *Releaser*: automatically maintains the file system's online disk cache at site-specified percentage usage thresholds by freeing disk blocks occupied by eligible archived files.
- *Stager*: restores file data to the disk cache. When a user or a process requests file data that has been released from disk cache, the stager automatically copies the file data back to the online disk cache.
- *Recycler*: clears archive volumes of expired archive copies and makes volumes available for reuse.

One of the key requirements from the outset of the transition to a new HSM was for the legacy UniTree data to remain transparently accessible to the user community through the new system. To facilitate the access of UniTree data on SAM-QFS, the entire file name space and directory structure of the UniTree system was recreated as directories and inodes in SAM-QFS. These inodes were basically placeholders, or links, to the original files in UniTree and contained an NCCS-defined volume serial number (VSN) and a "stranger" tape media type. Using the SAM migration toolkit, a set of libraries was created by Instrumental, Inc. to satisfy a stage request in SAM-QFS for a legacy UniTree file, which was identified to SAM by the "stranger" media type and the NCCS-defined VSN. Therefore, if a user requests a file that resides in UniTree, these libraries transparently retrieve the specified file from the UniTree system over a private network. Once the file has been retrieved from UniTree, it now exists within the SAM-QFS file system with two archive copies written to SAM tape and no longer needs to be retrieved from the legacy HSM.

Complementary to user driven access to the legacy data in UniTree, the NCCS has written Perl scripts to actively migrate the data from UniTree into SAM-QFS. These Perl scripts migrate files on a tape-by-tape basis and run "behind the scenes" to minimize the impact to the production environment. A single migration stream will secure files on a UniTree VSN from a well-defined list of UniTree tapes. This stream will get the current status of each file on that tape, i.e., whether or not the user has already migrated the file by retrieving it from tape or has even deleted the file. Next, the migration stream will begin to transfer the files over the private network using FTP. When the legacy files are retrieved to SAM-QFS disk cache, SAM writes two tape archive copies. After the migration stream has been completed, a separate analysis Perl script is run on each UniTree tape to verify that the files are in SAM-QFS. For quality control purposes, a checksum is run on every 100^{th} file. The current rate of migration is approximately 2 TB of data per day.

5. Conclusions

The integration effort of installing a new system to an existing High Performance Computing environment is difficult and requires much planning and effort. The

installation of the new Sun SAM-QFS system was no exception and many valuable lessons were learned.

- *Migration of Legacy Data*: The goal of migrating 100's of terabytes of data while still providing users with the ability to store and retrieve new files and transparently access legacy data is nontrivial and takes a significant number of resources. The amount of time and resources, i.e., tapes, tape drives, and network bandwidth, needs to be accurately estimated from the beginning and built into the integration plan such that users are not overly disrupted during the transition period as data is being migrated.
- *Test System*: It is important to have a test environment in which configuration modifications, such as operating system or storage software upgrades, may be tested without affecting the production environment.
- *User Account Management*: With two highly available domains on the new system, NCCS specific scripts were developed to synchronize user accounts between the two domains.
- *Pilot User Phase*: Before turning the system over to production computing, the internal NCCS staff and a set of pilot users were permitted access to the system. This phase allowed for a thorough testing of the environment before the full user community was allowed access.
- *Staff and User Training*: While the SAM-QFS system was designed to be as consistent as possible with UniTree, several training sessions were held with the staff and with the users to attempt to answer common questions. This allowed the user community to immediately begin using the new system and the staff to better support users from the beginning.
- *Software Upgrades*: Maintaining concurrency with the vendor's most recent release levels of operating systems and software is extremely important. Most vendors do not have the means to retroactively fix bugs for earlier release levels.
- *Security*: Define the necessary security requirements at the beginning of the process, and let those requirements drive the solution. It is more costly and disruptive to secure a system after it has been installed and patterns of use have developed by the user community.

The NCCS successfully transitioned the Sun SAM-QFS system into the production environment in September of 2003. The active migration of the more than 300 TB of data is slated to be completed in May of 2004. The new system has proven to be very reliable and capable of handling heavier loads than its predecessor. To date, the NCCS has seen tape activity, both user demand and migrations, exceed 9.8 TB for a single day.

As the NCCS continues to add computational capacity and as the user community continues to push the limits of modeling and assimilation to new heights, the HSM must evolve and adapt to the continued increase of requirements. The NCCS will incorporate the disk cache from UniTree into the production SAM-QFS system once the migration of the legacy data is complete. Also, the NCCS is analyzing the use of serial ATA, commodity based disk storage, as a second tier storage to sit between the high speed disk and slower tape. Finally, the NCCS is currently developing a data management system,

based on the Storage Resource Broker (SRB),[7] to provide user's with a single interface to storage and more control over their own data administration.

References

[1] http://nccs.nasa.gov.
[2] *Performance Management at an Earth Science Supercomputer Center*, Jim McGalliard and Dick Glassbrook.
[3] *Storage and Network Bandwidth Requirements Through the Year 2000 for the NASA Center for Computational Sciences,* Ellen Salmon, Proceedings of the fifth Goddard Conference on Mass Storage Systems and Technologies, (1996) pp. 273-286.
[4] *Mass Storage System Upgrades at the NASA Center for Computational Sciences,* A. Tarshish, E. Salmon, M. Macie, and M. Saletta, Proceedings of the Eight NASA Goddard Conference on Mass Storage Systems and Technologies, Seventh IEEE Symposium on Mass Storage Systems, (2000) pp. 325-334.
[4] *UniTree to SAM-QFS Project Plan*, Jeff Paffel, Instrumental, Inc., NCCS internal report.
[5] *UniTree to SAM-QFS Migration Procedure*, Daniel Duffy, Computer Sciences Corporation, NCCS internal report.
[6] *Sun SAM-FS and Sun SAM-QFS Storage and Archive Management Guide*, August 2002; *Sun QFS, Sun SAM-FS, and Sun SAM-QFS File System Administrator' s Guide.*
[7] http://www.npaci.edu/DICE/SRB/.

Parity Redundancy Strategies in a Large Scale Distributed Storage System

John A. Chandy
Dept. of Electrical and Computer Engineering
University of Connecticut
Storrs, CT 06269-1157
jchandy@uconn.edu
tel +1-860-486-5047

Abstract

With the deployment of larger and larger distributed storage systems, data reliability becomes more and more of a concern. In particular, redundancy techniques that may have been appropriate in small-scale storage systems and disk arrays may not be sufficient when applied to larger scale systems. We propose a new mechanism called delayed parity generation with active data replication (DPGADR) to maintain high reliability in a large scale distributed storage system without sacrificing fault-free performance.

1 Introduction

Data creation and consumption has increased significantly in recent years and studies have suggested that the amount of information stored digitally will continue to double every year for the foreseeable future. This increasing need for information storage leads to a corresponding need for high-performance and reliable storage systems. Single storage nodes can not provide the required storage capacity or scalability. Thus, in an effort to satisfy this need, there has been significant work in the area of distributed storage systems where storage nodes are aggregated together into a larger cohesive storage system. These include distributing data amongst shared disks [1, 6, 12], dedicated storage nodes [8], clustered servers [4], or the clients themselves [2, 7]. It is not unreasonable to expect systems with petabytes of data distributed across thousands of nodes in these distributed storage systems.

However, as we increase the number of nodes, the reliability of the entire system decreases correspondingly unless steps are taken to introduce some form of redundancy into the system. In this paper, we discuss redundancy mechanisms to provide high reliability without sacrificing fault free performance. For the purposes of this paper, we refer to individual storage devices in the distributed storage system as nodes - whether they be disks in a shared disk SAN, servers in a clustered server, or OBSDs in a network attached disk system. The techniques and strategies apply with slight variations to all implementations.

Data redundancy in most disk array subsystems is typically provided by using RAID. These same techniques used at the disk level can also be used at the node level. Mirroring,

or RAID1, entails replication of the data on multiple nodes. Parity striping, or RAID5, involves spreading data along with parity across multiple nodes. Choosing which RAID level to use is typically determined by cost and application requirements. At the disk array level, the redundancy choice is usually RAID5 as it provides excellent availability, little storage overhead, and adequate performance.

However, with a large scale distributed system, the choice is not so clear. RAID5 no longer provides sufficient reliability since a thousand node system could exhibit MTTFs of a few years. A mirrored distributed storage system, however, can have an MTTF of several decades. In addition, RAID5 suffers from the well-known write penalty whereby parity updates require two extra reads to generate the parity. Techniques to overcome this problem in a disk subsystem such as the use of non-volatile caches can not be used in a distributed system. Moreover, the cost of the write penalty is more significant because of the high latency costs inherent in network communications.

Because of the limitations of parity striping, in many distributed storage systems, replication or mirroring is the preferred strategy for redundancy [3, 15]. In addition, replication allows widely distributed clients and nodes to take advantage of locality and retrieve data from the closest storage node. However, the cost of mirroring is the 100% storage overhead.

In this paper, we present methods to achieve the low storage overhead of parity striping but retaining the performance and reliability characteristics of mirroring. In particular, we discuss the use of delayed parity generation to improve parity striping performance.

2 Delayed Parity Generation with Active Data Replication

The concept of delayed parity generation with active data replication (DPGADR) is based on reducing the number of accesses required to generate parity in a RAID5 system. In a standard RAID5 disk array, the array controller must read old values from both the data and parity disks and then write the new data back to the data disk and the XOR'ed parity result back to the parity disk. This results in a total of 4 disk accesses (potentially 2 if the parity and data reads had been cached). In a DPGADR system, we delay the generation of the parity, and thus do not require the reading of old data and parity or the writing the parity result. However, without parity generation, the system is potentially compromised in the event of failure. To address this, we replicate the new data to a *replication node* that is not part of the RAIDed redundancy group. We have reduced the number of accesses to just 2 writes - both of which can be proceed in parallel. Figure 1 shows the data distribution in a DPGADR system. In order to distribute the load, the replication node can be rotated across the redundancy group.

Each replication node keeps a map relating the actual block location to the active data locations kept on the node. Thus, the client is not responsible for identifying the block location for the replicated data on the replication node. The client can use a simple hash algorithm to map from block ID to replication node, and it is then the replication node's responsibility to allocate storage space locally.

On the surface, this DPGADR scheme appears to be simply mirroring of data. However, we do not maintain the mirrored data on the replication node in perpetuity. In order to avoid replicating all data writes, the replication node will periodically generate the parity for any

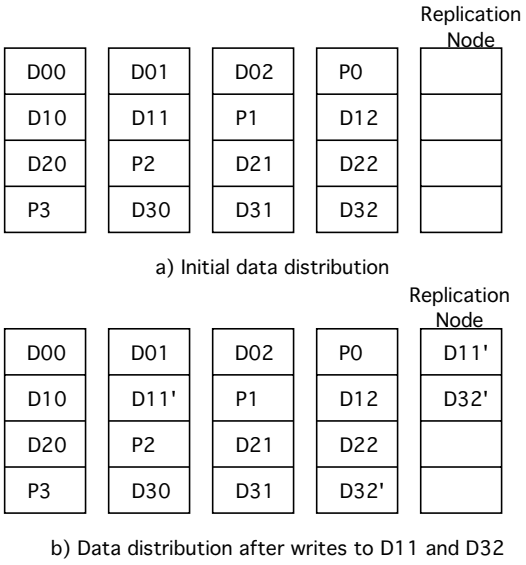


Figure 1: DPGADR data distribution.

blocks that it contains and then flush these blocks from its data store. Because of this periodic data flushing, the replication node is in effect a cache of actively used data blocks.

Parity generation from the replication node is not a trivial problem as the replication node may not have the required data to generate the parity. In such a case, the node must retrieve the stripe data from the relevant nodes before generating the parity. However, in general, it is likely that the active data set on the replication node will contain all or most of the data blocks in a particular stripe because of locality and small working set sizes [11]. We would like to keep the stripe length small so as to make sure that the entire stripe is in the active data set on the replication node. This is also desirable for reliability reasons since it reduces the size of the redundancy group. A reasonable stripe length is 5 nodes, thus requiring 200 stripes to span a 1000 node system.

3 Reliability

As mentioned above, as we increase the number of nodes in a large scale storage system, RAID5 parity striping no longer provides sufficient redundancy to give adequate system reliability. The use of DPGADR can improve system reliability significantly. Since recently used data is copied to a replication node, the system exhibits mean time to data loss (MTTDL) characteristics near to that of a mirrored system. Figure 2 illustrates how the system can tolerate more than one failure in a redundancy set and still recover the most recent data. Even though two nodes have failed, the active data blocks, *D11'* and *D32'*, are still available from the replication node. In fact if all the nodes except for the replication node fail, the DPGADR method allows for the recovery of all active data. While the replication node can prevent loss of active data in the presence of multiple failures, inactive data can still be lost if there is more than one fault. To prevent data loss in such a scenario, we require that the system have judicious backup procedures so that all inactive data is present on backup media. The replication node must be large enough to accommodate all active

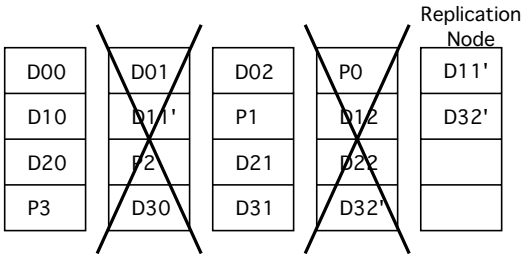


Figure 2: DPGADR failure scenario.

data between backups. This need not be that large as the working set size of a storage system over a 24 hour period is typically around 5% of the entire storage space [11]. Thus, it is sufficient to have one replication node for every 20 data nodes and using daily backups to prevent data loss of inactive data in the presence of dual faults. We also suspend access to the DPGADR group once a second fault has been recorded. This prevents invalid data being read from inactive data blocks. Thus, the DPGADR method prevents data loss in most dual fault cases but system availability is the same as a RAID system because of the access blocking due to a second fault.

We can develop a model for the MTTDL and availability of a DPGADR system using a similar analysis methodology to that outlined in [10]. The nodes are assumed to have independent and exponential failure rates. We assume d nodes per parity group, one redundancy node per n_G parity groups, and the mean time to failure of each node is $MTTF_{Node}$ The MTTDL for a DPGADR group is:

$$MTTDL_{DPGADR} = \frac{\frac{MTTF_{Node}}{n_G(d+1)+1}}{Pr[\text{data loss failure during repair time}]} \tag{1}$$

Data loss during the repair time of the failed node can happen in three cases: 1) the first failed node was the replication node and any other node fails, 2) the first failed node was not the replication node and the replication node fails, and 3) the failed node was not the replication node and two non-replication nodes in the same parity group fail. Thus, the probability of data loss causing failure during the repair time is as follows:

$$\begin{aligned} Pr[\text{data loss failure during repair time}] = \\ Pr[\text{first failed node was a replication node}]p_f \\ + \quad (1 - Pr[\text{first failed node was a replication node}])(p_{rf} + p_{2f}) \end{aligned} \tag{2}$$

where p_f is the probability that any one of the remaining $n_G(d+1)$ nodes fails during the repair time, p_{rf} is the probability that the replication node fails during the repair time, and p_{2f} is the probability that 2 non-replication nodes from the same parity group fail during the repair time. The derivation of p_f is straightforward. If we define the mean time to repair the node as $MTTR_{Node}$, then assuming exponential failure rates,

$$p_f \approx \frac{\frac{MTTR_{Node}}{n_G(d+1)}}{MTTF_{Node}} \tag{3}$$

Configuration	MTTDL (years)	Redundancy overhead
RAID5 (d=5)	7.9	200 nodes
Mirror	23.8	1000 nodes
DPGADR (d=5,n_G=4)	39.6	250 nodes

Table 1: MTTDL and Overhead for a 1000 data node system. ($MTTF_{Node}$ = 100000 hours and $MTTR_{Node}$ = 24 hours

when $MTTF_{Node} \gg MTTR_{Node}$. Similarly, p_{rf} is equal to $\frac{MTTR_{Node}}{MTTF_{Node}}$. p_{2f} can be expressed as follows:

$$p_{2f} = \binom{d}{2}(\frac{MTTR_{Node}}{MTTF_{Node}})^2(1-\frac{MTTR_{Node}}{MTTF_{Node}})^{d-1} \approx \frac{d(d-1)}{2}(\frac{MTTF_{Node}}{MTTR_{Node}})^2 \quad (4)$$

Substituting into Eqs. 1 and 2, we arrive at:

$$MTTDL_{DPGADR} = \frac{\frac{MTTF_{Node}}{n_G(d+1)+1}}{\frac{1}{n_G(d+1)+1}[\frac{MTTR_{Node}}{n_G(d+1)MTTF_{Node}}] + \frac{n_G(d+1)}{n_G(d+1)+1}[\frac{MTTR_{Node}}{MTTF_{Node}} + \frac{d(d-1)}{2}(\frac{MTTR_{Node}}{MTTF_{Node}})^2]}$$
$$\approx \frac{MTTF^2_{Node}}{n_G(d+1)MTTR_{Node}} \quad (5)$$

In a large system with n_S DPGADR groups, the $MTTDL$ is $\frac{MTTF^2_{Node}}{n_S n_G(d+1)MTTR_{Node}}$. If we define D as the total number of data nodes in the system, i.e. $n_S n_G d$, we can rewrite $MTTDL$ as $\frac{MTTF^2_{Node}}{D(1+\frac{1}{d})MTTR_{Node}}$. The redundancy overhead to support parity and replication is $\frac{D}{d} + \frac{D}{n_G\ d}$. By comparison, a mirrored system with D data nodes has a MTTDL of $\frac{MTTF^2_{Node}}{2D\ MTTR_{Node}}$ with an overhead of D nodes, and a RAID5 system has an MTTDL of $\frac{MTTF^2_{Node}}{D(d+1)\ MTTR_{Node}}$ and an overhead of $\frac{D}{d}$ nodes. The DPGADR system actually has better MTTDL times than a mirrored system with significantly less redundancy overhead. Table 1 shows MTTDL and overhead numbers for a 1000 data node system. Note that while the MTTDL of a DPGADR system is better than a mirrored system, the availability, i.e. the probability that the system is available for use, is more like a RAID5 system. This is because after a second failure, the system is suspended until the repair is complete.

4 Related Work

Xin et al [15], have proposed three different large storage system redundancy architectures with two fast recovery mechanisms: fast mirroring copy (FMC) and lazy parity backup (LPB). The LPB method is similar to the DPGADR scheme in that parity calculation is delayed. However, it relies on a RAID5 redundancy set to be completely mirrored at a

greater than 100% storage overhead. The DPGADR scheme requires significantly less overhead since only actively used data is replicated.

Other related work is in the area of RAID5 disk arrays and of particular interest are parity logging [13], data logging [5], and hot mirroring [9, 14].

The parity logging technique eliminates the need for parity disk accesses by caching the partial parity formed from the old and new data in non-volatile memory at the controller. The partial parity can then be periodically flushed to a log disk, which can then be cleaned out at a later time to generate the actual parity disk data. This process reduces the number of disk accesses from 4 to 2 and clearly, this reduction in accesses will greatly speed up the performance of writes in a RAID system. Parity logging, however, is not practical in a distributed system because of the need to cache data in non-volatile memory. It is not reasonable to expect distributed system clients to have non-volatile memory available. Moreover, the management of the cache across multiple clients can be problematic. The DPGADR system does not require non-volatility at the client since all data is pushed out to the replication node immediately. Data logging is similar to DPGADR except that it performs an old data read and stores that to the data log as well. This requires an extra disk access and the maintenance of log maps requires non-volatile memory at the clients as well.

Hot mirroring [9] and AutoRAID [14] are similar techniques that attempt to move actively used data to mirrored regions of the array and less frequently used data to parity logged regions (hot mirroring) or parity striped regions (AutoRAID). These system require a background process that evaluates the "hotness" of data and then moves them to or from mirrored regions as required. If a data block is in the parity striped region, it will remain there until a background process has tagged it as hot even if it is experiencing high activity. DPGADR systems, however, dynamically adjust to the activity of the data since the latest data is always pushed to the mirrored region, i.e. the replication node.

5 Conclusions

In this paper, we have described a delayed parity construction mechanism called DPGADR that allows parity striping to be used on large scale distributed storage systems without suffering from the small write performance penalty. Compared to mirroring it can reduce the storage overhead from 100% to less than 20% and compared to parity striping it can reduce small write accesses to just two parallel accesses. Because of redundancy in the active data replication node, the overall system reliability is better than mirroring for significantly less overhead.

References

[1] D. Anderson and J. Chase. Failure-atomic file access in the Slice interposed network storage system. *Cluster Computing*, 5(4):411–419, Oct. 2002.

[2] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless network file systems. In *Proceedings of the Symposium on Operating System Principles*, pages 109–126, Dec. 1995.

[3] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 34–43, June 2000.

[4] J. D. Bright and J. A. Chandy. A scalable architecture for clustered network attached storage. In *Proceedings of the IEEE/NASA Goddard Symposium on Mass Storage Systems and Technologies*, pages 196–206, Apr. 2003.

[5] E. Gabber and H. F. Korth. Data logging: A method for efficient data updates in constantly active RAIDs. In *Proceedings of the International Conference on Data Engineering*, pages 144–153, 1998.

[6] G. A. Gibson and R. Van Meter. Network attached storage architecture. *Commun. ACM*, 43(11):37–45, Nov. 2000.

[7] J. H. Hartman and J. K. Ousterhout. Zebra: A striped network file system. In *Proceedings of the USENIX 1992 Workshop on File Systems*, May 1992.

[8] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 84–92, Oct. 1996.

[9] K. Mogi and M. Kitsuregawa. Hot mirroring: A method of hiding parity update penalty and degradation during rebuilds for RAID5. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 183–194, June 1996.

[10] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 109–116, June 1988.

[11] C. Ruemmler and J. Wilkes. A trace-driven analysis of working set sizes. Technical Report HPL-OSR-93-23, Hewlett-Packard, Palo Alto, CA, Apr. 1993.

[12] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of USENIX Conference on File and Storage Technologies*, pages 231–244, Jan. 2002.

[13] D. Stodolsky, G. Gibson, and M. Holland. Parity logging: Overcoming the small write problem in redundant disk arrays. In *Proceedings of the International Symposium on Computer Architecture*, 1993.

[14] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, Feb. 1996.

[15] Q. Xin, E. L. Miller, T. Schwarz, D. D. E. Long, S. A. Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *Proceedings of the IEEE/NASA Goddard Symposium on Mass Storage Systems and Technologies*, pages 146–156, Apr. 2003.

Reducing Storage Management Costs via Informed User-Based Policies

Erez Zadok, Jeffrey Osborn, Ariye Shater, Charles Wright, and Kiran-Kumar Muniswamy-Reddy
Stony Brook University
{*ezk, jrosborn, ashater, cwright, kiran*}*@fsl.cs.sunysb.edu*

Jason Nieh
Columbia University
nieh@cs.columbia.edu

Abstract

Storage management costs continue to increase despite the decrease in hardware costs. We propose a system to reduce storage maintenance costs by reducing the amount of data backed up and reclaiming disk space using various methods (e.g., transparently compress old files). Our system also provides a rich set of policies. This allows administrators and users to select the appropriate methods for reclaiming space. Our performance evaluation shows that the overheads under normal use are negligible. We report space savings on modern systems ranging from 25% to 76%, which result in extending storage lifetimes by 72%.

1. Introduction

Despite seemingly endless increases in the amount of storage and decreasing hardware costs, managing storage is still expensive. Furthermore, backing up more data takes more time and uses more storage bandwidth—thus adversely affecting performance. Users continue to fill increasingly larger disks. In 1991, Baker reported that the size of large files had increased by ten times since the 1985 BSD study [1, 8]. In 2000, Roselli reported that large files were getting ten times larger than Baker reported [9]. Our recent studies show that just merely by 2003, large files are ten times larger than Roselli reported.

Today, management costs are five to ten times the cost of underlying hardware and are actually increasing as a proportion of cost because each administrator can only manage a limited amount of storage [4, 7]. We believe that reducing the rate of consumption of storage is the best solution to this problem. Independent studies [10] as well as ours indicate that significant savings are possible.

To improve storage management via efficient use of storage, we designed the *Elastic Quota System* (Equota). Elastic quotas enter users into a contract with the system: users can exceed their quota while space is available, under the condition that the system does not provide as rigid assurances about the file's safety. Users or applications may designate some files as *elastic*. Non-elastic (or persistent) files maintain existing semantics. Elastic quotas create a hierarchy of data's importance: the most important data will be backed up frequently; some data may be compressed and other data can be compressed in a lossy manner; and some files may not be backed up at all. Finally, if the system is running short on space, the elastic files may even be removed. Users and administrators can configure flexible policies to designate which files belong to which part of the hierarchy. Elastic quotas introduce little overhead for normal operations and demonstrate that through this new disk usage model, significant space savings are possible.

2. Motivational study

Storage needs are increasing—often as quickly as larger storage technologies are produced. Moreover, each upgrade is costly and carries with it high fixed costs [4]. We conducted a study to quantify this growth, with an eye toward reducing this rate of growth.

We identified four classes of files, three of which can reduce the growth rate and also the amount of data to be backed up. Similar classifications have been used previously [6] to reduce the amount of data to be backed up. First, there are files that cannot be considered for reducing growth. These files are important to users and should be backed up frequently, say daily. Second, studies indicate that 82–85% of storage is consumed by files that have not been accessed in more than a month [2]. Our studies confirm this trend: 89.1% of files or 90.4% of storage has not been accessed in the past month. These files can be compressed to recover space. They need not be backed up with the same frequency as the first class of files as least-

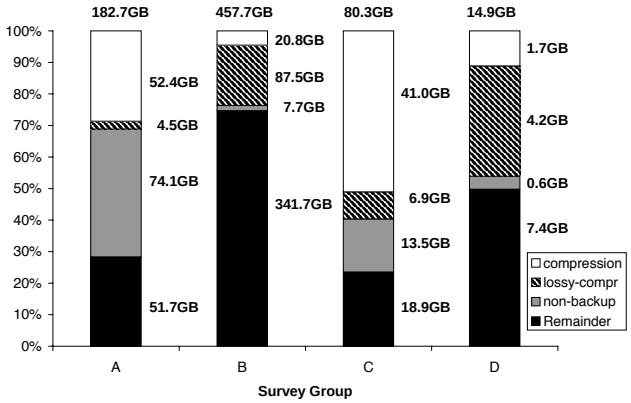


Figure 1. Space consumed by different classes. Actual amounts appear to the right of the bars, with the total size on top.

recently-used files are unlikely to change in the near future. Third, multimedia files such as JPEG or MP3 can be re-encoded with lower quality. This method carries some risk because not all of the original data is preserved, but the data is still available and useful. These files can be backed up less frequently than other files. Fourth, previous studies show that over 20% of all files—representing over half of the storage—are regenerable [10]. These files need not be backed up. Moreover, these files can be removed when space runs short.

To determine what savings are possible given the current usage of disk space, we conducted a study of four sites, to which we had complete access. These sites include a total of 3,898 users, over 9 million files, and 735.8GB of data dating back 15 years: (A) a small software development company with 100 programmers, management, sales, marketing, and administrative users with data from 1992–2003; (B) an academic department with 3,581 users, mostly students, using data from shared file servers, collected over 15 years; (C) a research group with 177 users and data from 2000–2003; and (D) a group of 40 cooperative users with personal Web sites and data from 2000–2003.

Each of these sites has experienced real costs associated with storage: A underwent several major storage upgrades in that period; B continuously upgrades several file servers every six months; the statistics for C were obtained from a file server that was recently upgraded; and D has recently installed quotas to rein in disk usage.

Figure 1 summarizes our study, starting with the top bar. We considered a transparent compression policy on all uncompressed files that have not been accessed in 90 days. We do not include already compressed data (e.g., `.gz`), compressed media (e.g., MP3 or JPEG), or files that are only one block long. In this situation, we save between 4.6% from group B to 51 % from group C. We yield large savings on group C: it has many `.c` files that compress well. Group B contains a large number of active users, so the percentage of files that were used in the past 90 days is less than that in the other sites. The next bar down (top hatched) is the savings from lossy compression of still images, videos, and sound files. The results varied from a savings of 2.5% for group A to a savings of 35% for group D. Groups B and D contain a large number of personal `.mp3` and `.avi` files. As media files grow in popularity and size, so will the savings from a lossy compression policy. The next bar down represents space consumed by regenerable files, such as `.o` files (with corresponding `.c`'s) and `~` files, respectively. This varied between 1.7% for group B to 40.5% for group A. This represents the amount of data that need not be backed up, or can be removed. Group A had large temporary backup tar files that were no longer needed. The amount of storage that cannot be reduced through these policies is the dark bar at the bottom. Overall, using the three space reclamation methods, we can save between 25% to 76.5% of the total disk space.

To verify if applying the aforementioned space reclamation methods would reduce the rate of disk space consumption, we correlated the average savings we obtained in the above environments with the SEER [5] and Roselli [9] traces. We require filename and path information, since our space reclamation methods depend on file types, which are highly correlated with names [3]. We evaluated several other traces, but only the combination of SEER and Roselli's traces provides us with the information we required. The SEER traces have pathname information but do not have file size information. Roselli's traces do not contain any file name information, but have the file size information. We used the size information obtained by Roselli to extrapolate the SEER growth rates. The Roselli traces were taken around the same time of the SEER traces, and therefore give us a good estimate of the average file size on a system at the time. At the rate of growth exhibited in the traces, the hard drives in the machines would need to be upgraded after 11.14 months. We observed that our policies extended the disks' lifetime to 19.2 months. The disk space growth rates were reduced by 52%. Based on these results, we have concluded that our policies offer promising storage management cost-reduction techniques.

3. Design

Our two primary design goals were to allow for versatile and efficient elastic quota policy management. To achieve versatility we designed a flexible policy configuration language for use by administrators and users. To achieve efficiency we designed the system to run as a kernel file system with a database, which associates user IDs, file names, and inode numbers. Our present implementation marks a file as

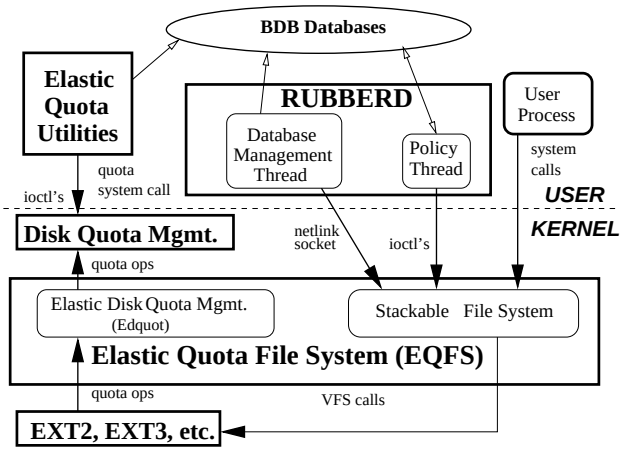


Figure 2. Elastic Quota Architecture

elastic using a single inode bit. A more complex hierarchy could be created using extended attributes.

Architecture Figure 2 shows the overall architecture of our system. There are four components in our system: (1) **EQFS** is a stackable file system that is mounted on top of another file system such as Ext3 [13]. EQFS includes a component (Edquot) that indirectly manages the kernel's native quota accounting. EQFS also sends messages to a user space component, *Rubberd*. (2) **Berkeley DB** (BDB) databases record information about elastic files [11]. We have two types of databases. First, for each user we maintain a database that maps inode numbers of elastic files to their names, allowing us to easily locate and enumerate each user's elastic files. The second type of database records an *abuse factor* for each user denoting how "good" or "bad" a given user has been with respect to historical utilization of disk space. (3) **Rubberd** is a user-level daemon that contains two threads. The database management thread is responsible for updating the BDB databases. The policy thread periodically executes cleaning policies. (4) **Elastic Quota Utilities** are enhanced quota utilities that maintain the BDB databases and control both persistent and elastic quotas.

System Operation EQFS intercepts file system operations, performs related elastic quota operations, and then passes the operation to the lower file system (e.g., Ext2). EQFS also intercepts the quota management system call and inserts its own set of quota management operations, *edquot*. Quota operations are intercepted in reverse (e.g., from Ext2 to the VFS), because only the native disk-based file system knows when an operation has resulted in a change in the consumption of inodes or disk blocks.

Each user on our system has two UIDs: one that accounts for persistent usage and another that accounts for elastic usage. The latter, called the *shadow UID*, is simply the ones-complement of the former. When an Edquot operation is called, Edquot determines if it was for an elastic or a persistent file, and informs dquot to account for the changed resource (inode or disk block) for either the UID or shadow UID. This allows us to use the existing quota infrastructure and utilities to account for elastic usage.

EQFS communicates information about creation, deletion, renames, hard links, and ownership changes of elastic files to Rubberd's database management thread over a *netlink* socket. Rubberd records this information in the BDB databases. Rubberd also records historical abuse factors for each user periodically, denoting the user's elastic space utilization over a period of time.

Elasticity Modes EQFS can determine a file's elasticity in five ways. (1) Users can explicitly toggle the file's elasticity, allowing them to control elasticity on a per file basis. (2) Users can toggle the elastic bit on a directory inode. Newly created files or sub-directories inherit the elastic bit. (3) Users can tell EQFS to create all new files elastically (or not). (4) Users can tell EQFS which newly-created files should be elastic by their extension. (5) Developers can mark files as elastic using two new flags we added to the `open` and `creat` system calls. These flags tell EQFS to create the new file as elastic or persistent.

4. Elastic quota policies

The core of the elastic quota system is its handling of space reclamation policies. File system management involves two parties: the running system and the people (administrators and users). To the system, file system reclamation must be efficient so as not to disturb normal operations. To the people involved, file system reclamation policies must consider three factors: fairness, convenience, and gaming. These three factors are important especially in light of efficiency as some policies can be executed more efficiently than others. We describe these three factors next.

Fairness Fairness is hard to quantify precisely. It is often perceived by the individual users as how they personally feel that the system and the administrators treat them. Nevertheless, it is important to provide a number of policies that could be tailored to the site's own needs. For example, some users might consider a largest-file-first compression or removal policy unfair because recently-created files may not remain on the system long enough to be used. For these reasons, we also provide policies that are based on individual users' disk space usage: users that consume more disk space over longer periods of time are considered the *worst offenders*. Once the worst offenders are determined and the

amount of disk space to clean from the users is calculated, the system must decide which specific files should be reclaimed from that user. Basic policies allow for time-based or size-based policies for each user. For the utmost in flexibility, users are allowed to define their own ordered list of files to be processed first.

Convenience For a system to be successful, it should be easy to use and simple to understand. Users should be able to find out how much disk space they are consuming in persistent and elastic files and which of their elastic files will be removed first. Administrators should be able to configure new policies easily. The algorithms used to define a worst offender should be simple and easy to understand. For example considering the current total elastic usage is simple and easy to understand. A more complex and fair algorithm could count the elastic space usage over time as a weighted average, although it might be more difficult for users to understand.

Gaming Gaming is defined as the ability of individual users to circumvent the system and prevent their files from being processed first. Good policies should be resistant to gaming. For example, a global LRU policy that compresses older files could be circumvented simply by reading those files. Policies that are difficult to circumvent include a per-user worst-offender policy. Regardless of the file's attributes, a user still owns the same total amount of data. Such policies work well on systems where it is expected that users will try to exploit the system.

4.1. Rubberd configuration files

When Rubberd has to reclaim space, it first determines how much space it should reclaim—the *goal*. The configuration file defines multiple policies, one per line. Rubberd then applies each policy in order until the goal is reached or no more policies can be applied. Each policy in this file has four parameters. (1) *type* defines what kind of policy to use and can have one of three values: `global` for a global policy, `user` for a per-user policy, and `user_profile` for a per-user policy that first considers the user's own personal policy file. (2) *method* defines how space should be reclaimed. Our prototype currently defines two policies: `gzip` compresses files and `rm` removes them. This allows administrators to define a system policy that first compresses files and then removes them if necessary. A policy using `mv` and `tar` could be used together as an HSM system, archiving and migrating files to slower media at cleaning time. (3) *sort* defines the order of files being reclaimed. We define several keys: `size` (in disk blocks) for sorting by largest file first, `mtime` for sorting by oldest modification time first, and similarly for `ctime` and `atime`. (4) *filter* is an optional list of file name filters to apply the policy to. If not specified, the policy applies to all files. If users define their own policy files and Rubberd cannot reclaim enough space, then Rubberd continues to reclaim space as defined in the system-wide policy file. HSM systems operate similarly, however, at a system-wide level [6].

4.2. Abuse factors

When Rubberd reclaims disk space, it must provide a fair mechanism to distribute the amount of reclaimed space among users. To decide how much disk space to reclaim from each user, Rubberd computes an *abuse factor* (AF) for all users. Rubberd then distributes the amount of space to reclaim from each user proportionally to their AF. We define two types of AF calculations: current usage and historical usage. Current usage can be calculated in three ways. First, Equota can consider the total elastic usage (in disk blocks) the user consumes. Second, it can consider the total elastic usage minus the user's available persistent space. Third, Equota can consider the total amount of space consumed by the user (elastic and persistent). These three modes give a system administrator enough flexibility to calculate the abuse fairly given any group of users (we also have modes based on a percentage of quota). Historical usage can be calculated either as a linear or as an exponential average of a user's disk consumption over a period of time (using the same metrics as current usage). The linear method calculates a user's abuse factor as the linear average over time, whereas the exponential method calculates the user's abuse with an exponentially decaying average.

4.3. Cleaning operation

To reclaim elastic space, Rubberd periodically wakes up and performs a `statfs` to determine if the high watermark has been reached. If so, Rubberd spawns a new thread to perform the reclamation. The thread reads the global policy file and applies each policy sequentially, until the low watermark is met or all policy entries are applied.

The application of each policy proceeds in three phases: abuse calculation, candidate selection, and application. For user policies, Rubberd retrieves the abuse factor of each user and then determines the number of blocks to clean from each user proportionally to the abuse factor. For global policies this step is skipped since all files are considered without regard to the owner's abuse factor. Rubberd performs the candidate selection and application phases only once for global policies. For user policies these two phases are performed once for each user. Rubberd then gets the attributes (size and times) for each file (EQFS allows Rubberd to get these attributes more efficiently by inode number rather than by name as required by `stat`). Rub-

berd then sorts the candidates based on the policy (e.g., largest or oldest files first). In the application phase, we reclaim disk space (e.g., compress the file) from the sorted candidates. Cleaning terminates once enough space has been reclaimed.

5. Related work

Elastic quotas are complementary to HSM systems. HSM systems provide disk backup as well as ways to reclaim disk space by moving less-frequently accessed files to a slower disk or tape. These systems then provide a way to access files stored on the slower media, ranging from file search software to replacing the migrated file with a link to its new location. Several HSM systems are in use today including UniTree, SGI DMF (Data Migration Facility), the SmartStor Infinet system, IBM Storage Management, Veritas NetBackup Storage Migrator, and parts of IBM OS/400. HP AutoRaid migrates data blocks using policies based on access frequency [12]. Wilkes et. al. implemented this at the block level, and suggested that per-file policies in the file system might allow for more powerful policies; however, they claim that it is difficult to provide an HSM at the file system level because there are too many different file system implementations deployed. We believe that using stackable file systems can mitigate this concern, as they are relatively portable [13]. In addition, HSMs typically do not take disk space usage per user over time into consideration, and users are not given enough flexibility in choosing storage control policies. We believe that integrating user- and application-specific knowledge into an HSM system would reduce overall storage management costs significantly.

6. Conclusions

The main contribution of this paper is in the exploration and evaluation of various elastic quota policies. These policies allow administrators to reduce the overall amount of storage consumed and to control what files are backed up when, thereby reducing overall backup and storage costs. Our system includes many features that allow both site administrators and users to tailor their elastic quota policies to their needs. Through the concept of an abuse factor we have introduced historical use into quota systems. Finally, our work provides an extensible framework for new or custom policies to be added.

We evaluated our Linux prototype extensively. Performance overheads are small and acceptable for day-to-day use. We observed an overhead of 1.5% when compiling `gcc`. For a worst-case benchmark, creation and deletion of empty files, our overhead is 5.3% without database operations (a mode that is useful when recursive scans may already be performed by backup software) and as much as 89.9% with optional database operations. A full version of this paper, including a more detailed design and a performance evaluation, is available at `www.fsl.cs.sunysb.edu/docs/equota-policy/policy.pdf`.

References

[1] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a Distributed File System. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 198–212. Association for Computing Machinery SIGOPS, 1991.

[2] J. M. Bennett, M. A. Bauer, and D. Kinchlea. Characteristics of files in NFS environments. *ACM SIGSMALL/PC Notes*, 18(3-4):18–25, 1992.

[3] D. Ellard, J. Ledlie, and M. Seltzer. The Utility of File Names. Technical Report TR-05-03, Computer Science Group, Harvard University, March 2003.

[4] Gartner, Inc. Server Storage and RAID Worldwide. Technical report, Gartner Group/Dataquest, 1999. `www.gartner.com`.

[5] G. H. Kuenning. *Seer: Predictive File Hoarding for Disconnected Mobile Operation*. PhD thesis, University of California, Los Angeles, May 1997.

[6] Julie Lugar. Hierarchical storage management (HSM) solutions today. `http://www.serverworldmagazine.com/webpapers/2000/10_camino.shtml`, October 2000.

[7] J. Moad. The Real Cost of Storage. *eWeek*, October 2001. `www.eweek.com/article2/0,4149,1249622,00.asp`.

[8] J. Ousterhout, H. Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In *Proceedings of the 10th ACM Symposium on Operating System Principles*, pages 15–24, Orcas Island, WA, December 1985. ACM.

[9] D. Roselli, J. R. Lorch, and T. E. Anderson. A Comparison of File System Workloads. In *Proc. of the Annual USENIX Technical Conference*, pages 41–54, June 2000.

[10] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R.W. Carton, and J. Ofir. Deciding When to Forget in the Elephant File System. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 110–123, December 1999.

[11] M. Seltzer and O. Yigit. A new hashing package for UNIX. In *Proceedings of the Winter USENIX Technical Conference*, pages 173–84, January 1991. `www.sleepycat.com`.

[12] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID Hierarchical Storage System. In *ACM Transactions on Computer Systems*, volume 14, pages 108–136, February 1996.

[13] E. Zadok and J. Nieh. FiST: A Language for Stackable File Systems. In *Proceedings of the Annual USENIX Technical Conference*, pages 55–70, June 2000.

A DESIGN OF METADATA SERVER CLUSTER IN LARGE DISTRIBUTED OBJECT-BASED STORAGE

Jie Yan, Yao-Long Zhu, Hui Xiong, Renuga Kanagavelu, Feng Zhou, So LihWeon
Data Storage Institute, DSI building, 5 Engineering Drive 1, Singapore 117608
{Yan_jie, Zhu_Yaolong}@dsi.a-star.edu.sg
tel +65-68748085

Abstract
In large distributed Object-based Storage Systems, the performance, availability and scalability of the Metadata Server (MDS) cluster are critical. Traditional MDS cluster suffers from frequent metadata access and metadata movement within the cluster. In this paper, we present a new method called Hashing Partition (HAP) for MDS cluster design to avoid these overheads. We also demonstrate a design using HAP to achieve good performance of MDS cluster load balancing, failover and scalability.

1. Introduction

Unlike traditional file storage systems with metadata and data managed by the same machine and stored on the same device [1], the object-based storage system separates the data and metadata management. An Object-based Storage Device (OSD) [2] cluster manages low-level storage tasks such as object-to-block mapping and request scheduling, and presents an object access interface instead of block-level interface [3]. A separate cluster of MDS manages metadata and file-to-object mapping, as shown in Figure 1. The goal of such storage system with specialized metadata management is to efficiently manage metadata and improve the overall system performance. In this paper, we mainly address performance, availability and scalability issues for the design of MDS cluster in Object-based Storage Systems.

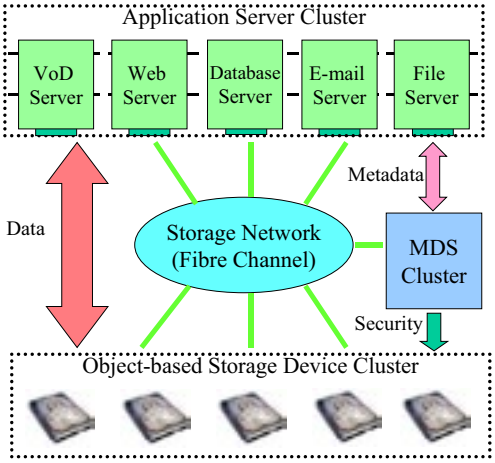


Figure 1. Object-based Storage System

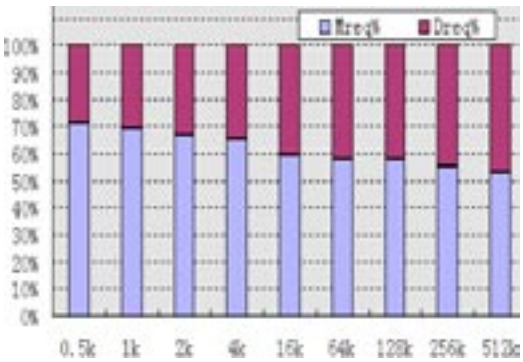


Figure 2 shows the data request percent (Dreq%) and the metadata request percent (Mreq%) of the total requests. This test is based on our OSD prototype (one client, one MDS and one OSD) connected by Fibre Channel, using Postmark (1000 files, 10 subdirectories, random access, 500 transactions).

Two key concerns about MDS cluster are the request load of metadata and load balancing within the cluster. In our preliminary OSD prototype, which adopts the traditional directory sub-tree to manage metadata, we find that more than 70 percent of all file system access requests are for metadata when using Postmark [4] to access 0.5k files, as shown in Figure 2. Although

the size of the metadata is generally small compared to the overall storage capacity, the traffic volume of such metadata access degrades the system performance. The large number of metadata requests can be attributed to the use of directory sub-tree metadata management.

Apart from metadata requests, an uneven load distribution within a MDS cluster would also raise severe bottleneck. Based on traditional cluster architecture, the performance of the load balancing, failover and scalability in the MDS cluster is limited, because most of these operations lead to the inevitable massive metadata movement within cluster. The Lazy Hybrid metadata management method [5] presented a hashing metadata management with the hierarchical directory support, which dramatically reduced the total number of metadata requests, but Lazy Hybrid did not deal with reducing metadata movement between MDSs for load balancing, failover and scalability.

This paper presents the new method called Hashing Partition (HAP) for MDS cluster design. HAP herein also adopts the hashing method, but focuses on reducing the cross MDS metadata movement in a clustered design, in order to achieve high performance of load balancing, failover and scalability.

The rest of the paper is organized as follows. The next section details the design of HAP and section 3 demonstrates our solutions of MDS Cluster load balancing, failover and scalability. Section 4 discusses MDS Cluster Rebuild. Finally, the conclusion of the paper is drawn in section 5.

2. Hashing Partition

Hashing Partition (HAP) provides a total solution for the file hashing, metadata partitioning, and metadata storage. There are three logical modules in the HAP: file

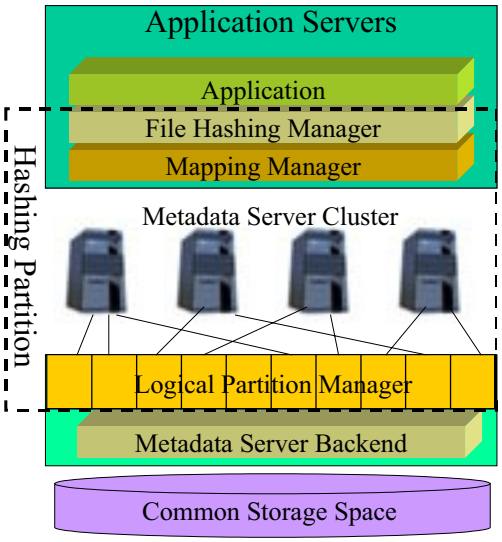


Figure 3. Hashing Partition

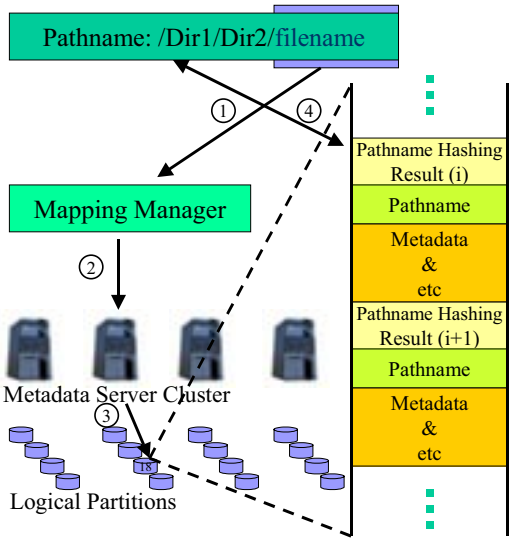


Figure 4. Metadata Access Pattern
①.Filename hashing, ②.Selecting MDS through Mapping Manager, ③.Accessing metadata by pathname hashing result, ④.Returning metadata to application server.

hashing manager, mapping manager, and logical partition manager, as shown in Figure 3.

In addition, HAP employs an independent common storage space for all MDSs to store metadata, and this space is divided into multiple logical partitions. Each logical partition contains part of global metadata table. Each MDS can mount and then exclusively access logical partitions allocated to it. Thus as a whole, MDS cluster can access a unique global metadata table.

The procedure of metadata access is described as follows. Firstly, file hashing manager hashes a filename to an integer, which can be mapped to the partition that stores the metadata of the file. Secondly, mapping manager can figure out the identity number of MDS that currently mounts that partition. Then client can send metadata request with the hash value of pathname to the MDS. Finally, logical partition manager located in MDS side accesses metadata on the logical partition in the common storage space. Figure 4 describes this efficient metadata access procedure. Normally, only a single message to a single metadata server is required to access a file's metadata.

2.1. File Hashing Manager

File hashing manager performs two kinds of hashing: filename hashing for partitioning metadata in MDS cluster, and pathname hashing for the metadata allocation in MDS. To access metadata of a file in MDS cluster, client needs to know two facts: which MDS manages the metadata and where the metadata is located in the logical partition. Filename hashing answers the first question and pathname hashing solves the second one. For example, if the client needs to access the file, "/a/b/filec", client uses the hashing result of "filec" to select MDS that manages the metadata. Then instead of accessing directory "a" and "b" to know where is the metadata of "filec", a hash result of "/a/b/filec", directly indicates where to retrieve the metadata.

But the filename hashing may introduce a potential bottleneck when a large parallel access to different files with the same name in different directories. Fortunately, the different hash values of various popular filenames, such as *readme* and *makefile*, make all these "hot points" distributed among MDS cluster and reduce the possibility of the potential bottleneck. In addition, even if certain MDS is over-loaded, our dynamic load balancing policy (section 3.1) can effectively handle this scenario and shift the "hot points" from overloaded MDS to the less-loaded MDSs.

2.2. Logical Partition Manager

Logical partition manager manages all logical partitions in the common storage space. It performs many logical partition management tasks, e.g. *mount*/*un-mount, backup* and *Journal recovery*. For instance, logical partition manager can periodically backup logical partitions to a remote backup server.

2.3. Mapping Manager

Mapping manager performs two kinds of mapping tasks: hashing result to logical partition mapping and logical partition to MDS mapping. Equation 1 describes these two mapping functions.

$$
\begin{aligned}
&Pi = f(H(filename)) \\
&MDSi = ML(Pi, PWi, MWi) \\
&Pi \in \{0, Pn\}; H(filename) \in \{0, Hn\}; MDSi \in \{0, Mn\} \\
&(Hn \geq Pn \geq Mn > 0)
\end{aligned} \tag{1}
$$

Where, *H* represents a filename hashing function; *f* stands for the mapping function that transfers hashing result to partition number (*Pi*); *ML* represents the function that figures out MDS number (*MDSi*) from partition number and related parameters (*PW* and *MW* will be explained in section 3.1); *Pn* is the total number of partitions; *Hn* is the maximum hashing value and *Mn* is the total number of MDSs.

When *PW* and *MW* are set, mapping manager simplifies the mapping function *ML* to a mapping table MLT, which describes the current mapping between MDS and logical partition. It is noted that one MDS can mount multiple partitions, but one partition can only be mounted to one MDS. To access metadata, mapping manager can indicate the logical partition that stores the metadata of a file based on the hash result of the filename. Then through MLT, mapping manager knows which MDS mounts that partition and manages the metadata of the file. Finally the client contacts the selected metadata server to obtain the file's metadata, file-to-object mapping and security information. Table 1 gives an example of MLT. Based on this table, in order to access metadata on logical partition 18, client needs to send request to *MDS1*.

Table 1. Example of MLT

Logical partition Number	MDS ID	MDS Weight
0~15	0	300
16~31	1	300
32~47	2	300
48~63	3	300

3. Load Balancing, Failover and Scalability

3.1. MDS Cluster Load Balancing Design

We propose a simple Dynamic Weight algorithm to dynamically balance the load of MDSs. HAP assigns a MDS Weight (*MW*) to each MDS according to its CPU power, memory size and bandwidth, and uses a Partition Weight (*PW*) to reflect the access frequency of each partition. *MW* is a stable value if the hardware configuration of the MDS cluster does not change, and *PW* can be dynamically adjusted according to the access rate and pattern of partitions. In order to balance the load between MDSs, mapping manager allocates partitions to MDS based on Equation 2.

$$
\frac{\sum PWi}{MWi} = \frac{\sum_{a=0}^{Pn} PWa}{\sum_{a=0}^{Mn} MWa} \tag{2}
$$

Where, $\sum PWi$ presents the sum of *PW* of all partitions mounted by *MDSi*; *Pn* stands for the total number of partitions; *Mn* presents the total number of MDSs.

In addition, each MDS needs to maintain load information about itself and all partitions mounted on it, and periodically uses Equation 3 to calculate new values.

$$MDSLOAD(i+1) = MDSLOAD(i) \times \alpha\% + MDSCURLOAD \times (1-\alpha\%) \quad (3)$$
$$PLOAD(i+1) = PLOAD(i) \times \beta\% + PCURLOAD \times (1-\beta\%)$$

Where, *MDSCURLOAD* is the current load of the MDS; *PCURLOAD* is the current load of the logical partition; *MDSLOAD(i)* represents the load status of a MDS at time *i*; *PLOAD(i)* stands for the load status of a logical partition at time *i*; α and β are constant used to balance the effects of old value and new value.

However, MDSs don't report their load information to the master node, e.g. one particular MDS, until a MDS alarms in its overloaded situation, such as the MDSLOAD exceeding the preset maximum load of the MDS. After receiving load information from all MDSs, the master node sets the *PW* using new *PLOAD* values. Then according to new *PW* and Equation 2, HAP shifts the control of certain partitions from the over-loaded MDS to some less-loaded MDSs and modifies MLT accordingly. This adjustment does not involve any physical metadata movement between MDSs.

3.2. MDS Cluster Failover Design

Typically, a conventional failover design adopts a standby server to take over all services of the failed server. In our design, the failover strategy relies on the clustered approach. In the case of a MDS failure, mapping manager assigns other MDSs to take over the work of the failed MDS based on Equation 2. Then the logical partition manager allocates the logical partitions managed by the failed MDS to its successors, as shown in Figure 5. So application servers can still access metadata on the same logical partition in the common storage space through the successors.

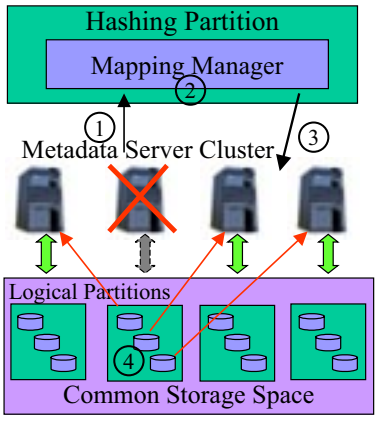


Figure 5. MDS cluster failover procedure
①.Detecting the MDS failure, ②.Recalculating MW and adjusting MLT, ③.Other MDSs take over logical partitions of the failure one, ④.Journal recovery

3.3. MDS Cluster Scalability Design

HAP significantly simplifies the procedure to scale the metadata servers. If the current MDS cluster cannot handle metadata request effectively due to the heavy load, new MDSs can be dynamically set up to release the overhead of others. HAP method allows the addition of MDS by adjusting *MWs* and thus generating a new MLT based on *ML*. This process doesn't touch the mapping relationship between filename and logical partition, because the number of logical partitions is unchanged. Following the new MLT, logical partition manager un-mounts certain partitions from existing MDSs and mounts them to the new MDS. This procedure also doesn't introduce any physical metadata movement within MDS cluster.

4. MDS Cluster Rebuild

Although HAP method can dramatically simplify the operation of MDS addition and removal, HAP actually has a scalability limitation, called Scalability Capability. The

preset number of logical partitions limits Scalability Capability, since one partition can only be mounted and accessed by one MDS at a time. For instance 64 logical partitions can only support up to 64 MDSs without rebuild. In order to improve Scalability Capability, we can add storage hardware to create new logical partitions and redistribute metadata among the entire cluster. This metadata redistribution introduces multi-MDS communication because the change in the number of logical partitions requires a new mapping function *f* in Equation 1, and affects the metadata location of the existing files in logical partitions. For example, after Scalability Capability is improved from 64 to 256, the metadata of a file may need to move from logical partition 18 to logical partition 74. The procedure that redistributes all metadata based on new mapping policy and improves Scalability Capability, is called MDS Cluster Rebuild.

In order to reduce the response time of MDS cluster rebuild, HAP adopts Deferred Update algorithm, which defers metadata movement and distributes its overhead. After receiving the cluster rebuild request, HAP saves a copy of the mapping function *f*, creates a new *f* based on the new number of logical partitions, and generates a new MLT. Then logical partition manager mounts all logical partitions including both the old and new according to the new MLT. After that, HAP responses immediately to the rebuild request and changes MDS cluster to a rebuild mode. Thus the initial operation for this entire process is very fast.

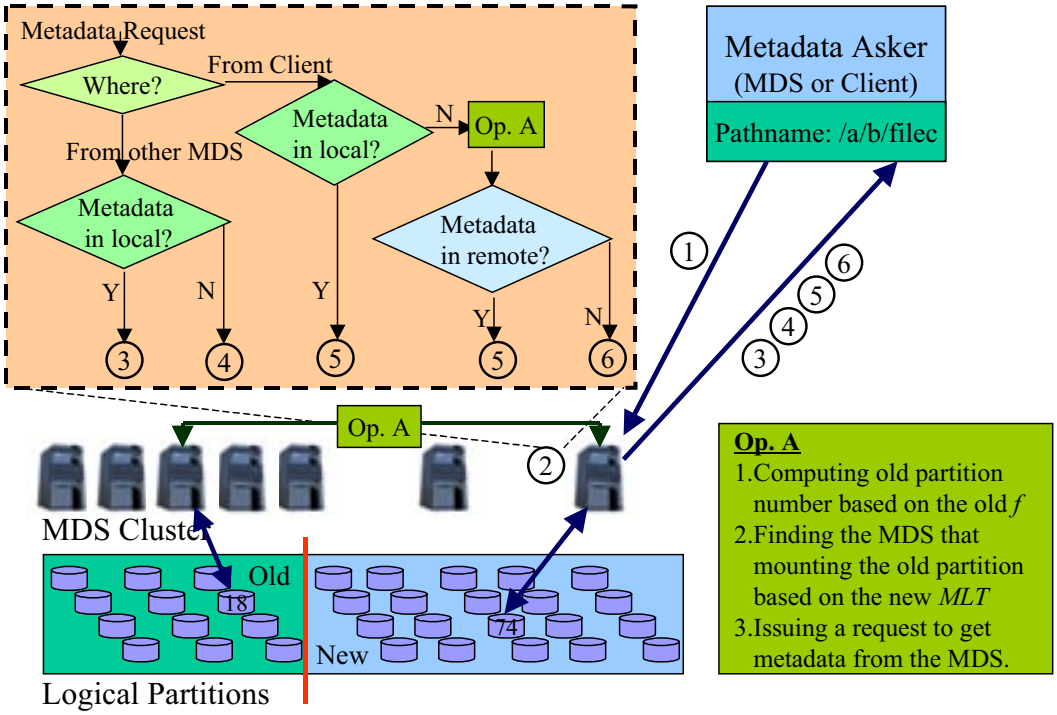


Figure 6. MDS Cluster Rebuild

①.Sending request to MDS based on new mapping result, ②.Searching for metadata and making judgment (the rectangle on the left shows the internal logic and Op. A is explained in the bottom rectangle), ③.Returning metadata and deleting it in local, ④.Reporting Error, ⑤.Returning metadata, ⑥.Wrong filename

During the rebuild, the behavior of the system is as if all the metadata had been moved to the right logical partitions. Actually, HAP updates or moves the metadata upon the first access. If a MDS receives a metadata request, and the metadata hasn't been moved to the logical partition that is mounted by it, the MDS needs to use the old mapping function *f* to calculate the original logical partition number based on the filename. Then through the new MLT, the MDS can find the MDS that currently mounts the original logical partition and send a metadata request to it. So the MDS can retrieve the metadata and complete the metadata movement. Figure 6 describes this procedure in detail. In addition, in order to accelerate the metadata movement progress, HAP can also adopt an independent thread to travel the metadata database and move the affected metadata only during the spare time of system.

5. Conclusion

We present a new method of Hashing Partition to manage metadata server cluster in large distributed object-based storage system. We use hashing method to avoid the numerous metadata accesses, and use filename hashing policy to remove the overhead of multiple MDS communication. Furthermore, based on the concept of logical partitions in the common storage space, HAP method significantly simplifies the implementation of the MDS cluster and provides efficient solutions for load balancing, failover and scalability.

The design described in this paper is part of our BrainStor project that targets to provide the full object-based storage solution. Currently we are implementing the Hashing Partition management for MDS Cluster in the BrainStor prototype. We also plan to explore the application of BrainStor technologies in Grid storage.

References

[1] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith. "Andrew: A distributed personal computing environment", *Communications of the ACM*, 29(3):184–201, Mar. 1986.

[2] R. O. Weber. "Information technology—SCSI object-based storage device commands (OSD)", Technical Council Proposal Document T10/1355-D, Technical Committee T10, Aug. 2003.

[3] Thomas M. Ruwart, "OSD: A Tutorial on Object Storage Devices", *19th IEEE Symposium on Mass Storage Systems and Technologies*, University of Maryland, Maryland, USA, April 2002.

[4] KATCHER, J. "Postmark: A new file system benchmark", Tech. Rep. TR3022 (Oct. 1997). Network Appliance.

[5] Scott A. Brandt, Lan Xue, Ethan L. Miller, and Darrell D. E. Long. "Efficient metadata management in large distributed file systems", *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 290–298, April 2003.

AN ISCSI DESIGN AND IMPLEMENTATION

Hui Xiong, Renuga Kanagavelu, Yaolong Zhu, and Khai Leong Yong
Data Storage Institute,
DSI Building, 5 Engineering Drive 1,
Singapore 117608
Tel:+65-68748100, Fax: +65-68732745
{Xiong_Hui, Renuga_KANAGAVELU, ZHU_Yaolong, YONG_Khai_Leong}
@dsi.a_star.edu.sg

Abstract
iSCSI is a network storage technology designed to provide an economical solution over a TCP/IP Network. This paper presents a new iSCSI design with multiple TCP/IP connections. The prototype is developed and experiments are conducted for performance evaluation on Gigabit Ethernet (GE). Test results show that the new iSCSI design improves performance 20%~60% compared with normal iSCSI architecture. Throughput can reach 107MB/s for big I/O and I/O rate can reach 15000 IOPS for small I/O.

1. Introduction

iSCSI is a network storage technology which transports SCSI commands over TCP/IP network. It has attracted a lot of attention due to the following advantages:

a. Good scalability. iSCSI is based on SCSI protocol and TCP/IP network, which can provide good scalability.
b. Low-cost. iSCSI can share and be compatible with existing TCP/IP networks. The user does not need to add any new hardware.
c. Remote data transferring capability. TCP/IP network can extend to metro area, which makes iSCSI suitable for remote backup and disaster recovery applications.

Most current iSCSI implementations use software solutions in which iSCSI device drivers are added on top of the TCP/IP layer for off-the-shelf network interface cards (NICs). It may cause performance issue. Many iSCSI researches and projects have been carried out to analyze the problem. A prior research project of iSCSI – Netstation project of USC showed that it was possible for iSCSI to achieve the 80% performance of direct-attached SCSI device [1]. IBM Haifa Research Lab carried out research on the design and the performance analysis of iSCSI [2, 3]. Bell Laboratories also did some test and performance study of iSCSI over metro network [4]. Some of solutions have also been brought forward to handle the performance issue. A research group proposed a solution to use memory of iSCSI initiator to cache iSCSI data [5]. Other solutions included using a TCP/IP offload Engine (TOE) [6] and even iSCSI adapter [7] to reduce

the burden of host CPU by offloading the processing of TCP/IP and iSCSI protocol into the hardware on the network adapter. But these hardware solutions will add the extra cost compared to a software solution.

The improvement of semiconductor technology has led to the rapid increase of CPU speed and memory access speed. During the past two years, commercial CPU speed has almost tripled from 1GHz to 3GHz, and memory bandwidth has also doubled from around 200~300MB/s to around 400~500MB/s. A powerful hardware platform makes it possible to achieve good performance for iSCSI software solutions.

The paper presents a new software iSCSI design and implementation, which employs multiple TCP/IP connections. Experiments are conducted on the GE network both in the lab and metro network environment. Testing results are analyzed and discussed.

2. Software iSCSI Design

2.1 iSCSI Storage Architecture

Figure 1 shows an iSCSI storage architecture including an initiator and a target, which communicate to each other via the iSCSI protocol. In the initiator, the application, which needs to store and access data to/from the storage device, issues file requests. The file system converts file requests to block requests from application to block device layer and SCSI layer. The initiator iSCSI driver encapsulates SCSI commands in iSCSI Protocol Data Units (PDUs) and sends them to the Ethernet network via the TCP/IP layer. The target iSCSI driver receives iSCSI PDUs from TCP/IP layer and de-capsulates it. Then SCSI commands are mapped to RAM, called RAM I/O, or mapped to an actual magnetic storage disk, called DISK I/O. The target driver then sends response data and status back to the TCP/IP layer. The low-level flow control of iSCSI fully follows the

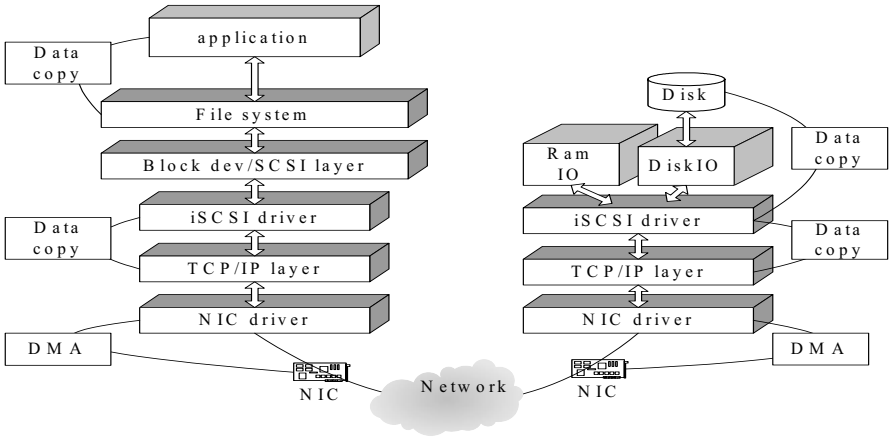


Fig.1 General iSCSI architecture

TCP/IP and Ethernet communication mechanism, which connects initiator and target by IP network. It is noted that two data copies and one DMA are processed in the initiator side during one IO access. For RAM I/O, one data copy and one DMA are processed in target.

2.2 A new iSCSI Design and Implementation

Normal iSCSI implementation is based on the single TCP/IP connection, which may not sufficiently utilize the network bandwidth. Furthermore, it may face serious performance issue caused by packets loss and long latency in metro network environment. We propose a new iSCSI architecture with multiple TCP/IP connections. This new architecture actually employs multiple virtual connections over one physical Ethernet connection (by one NIC), which is different from general idea of multiple Ethernet physical connections (by multiple NICs) according to iSCSI Request for Commands (iSCSI RFC) documents. The new design supposes not only to improve the iSCSI performance by increasing the utilization of network bandwidth, but also to provide a better mechanism to handle the long latency issue in metro network environment.

The working principle of our new design is shown in Figure 2. Multiple virtual TCP/IP connections are built on one physical Ethernet connection. One connection is used for sending SCSI request from initiator to target; another one is used for sending response from target to initiator. One pair of transmitting thread (Tx_thread) and receiving thread (Rx_thread), which locate in initiator or target respectively, are responsible for data communication within one connection.

We use the read operation as an example to explain the detailed communication procedure. The SCSI middle layer is a standard interface for SCSI layer to communicate with iSCSI device driver. The two main functions of SCSI middle layer are queuecommand( ) which issues SCSI command to the iSCSI driver and done( ) which informs SCSI middle layer that the command is finished by iSCSI driver. The iSCSI

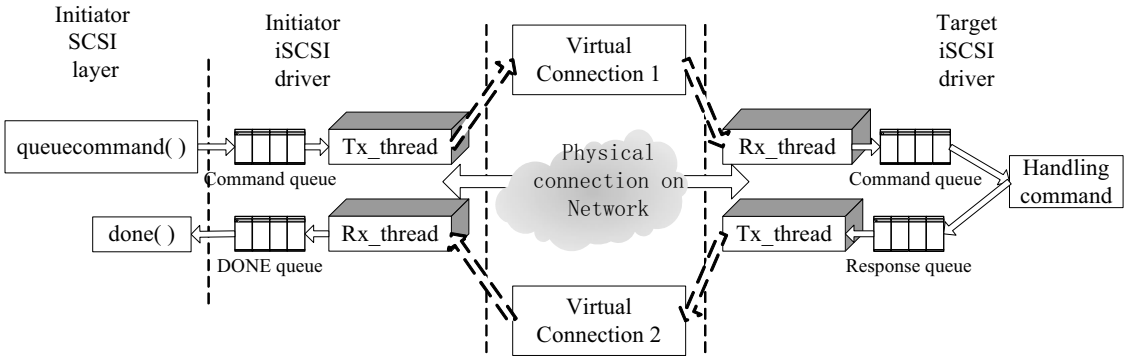


Fig 2. Multiple parallel connections iSCSI architecture

initiator driver gets read commands from SCSI middle layer. These commands are encapsulated in iSCSI request PDUs and then queued in the initiator command-queue. As shown in Figure 2, The Tx_thread of connection 1 sends these PDUs from command-queue to the target. The target driver receives these PDUs by the Rx_thread of the same connection. After de-encapsulating iSCSI request PDUs and mapping SCSI commands to the storage device, the target driver gets data from storage device and forms iSCSI response PDUs (including data and status). Then these iSCSI response PDUs are queued in the response-queue. Tx_thread in the target sends these PDUs by connection 2. The initiator driver receives response through the Rx_thread of the connection 2. Then the initiator driver put it to done-queue to call the done( ) function to finish the SCSI exchange.

As multiple connections are used, synchronization becomes important. According to iSCSI draft, the "initiator task tag" is used to record the sequence number of the iSCSI command in every iSCSI PDU. Related data and Status PDUs of the iSCSI command attach the same "initiator task tag". Even if multiple connections are used and command queue is enabled in both initiator and target, the device driver can easily find respective data/status PDUs from the pending queue.

3. Experiments

Figure 3 shows the system configuration used in our iSCSI experiments in the lab environment. One Nera-summit-5i Gigabit Ethernet (GE) switch (supporting 9K jumbo frames) is used to connect iSCSI initiator and target. One Finisar GTX THG iSCSI analyzer is used to monitor and capture all Ethernet packets for detailed analysis. Further experiments are conducted in the metro network environment. iSCSI initiator and target are connected through fiber with the aid of DWDM switch over 25KM physical distance, as shown in Figure 4.

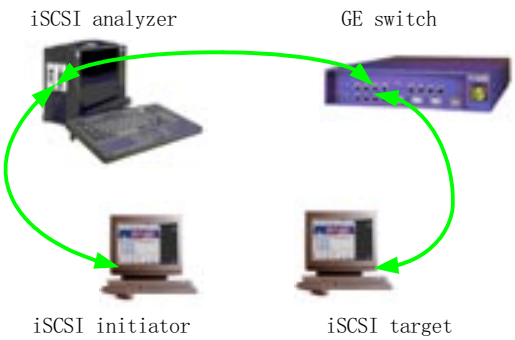


Fig. 3 Experiment platform in Lab

Fig.4 Metro network experiment

The hardware configuration of the iSCSI target includes a 64-bit/133MHz PCI-X motherboard with 2.4GHz P4 processor and 512M RAM. The iSCSI initiator is implemented on a 64-bit/66MHz PCI motherboard with 2.4GHz P4 processor and 256MB RAM. Intel PRO1000F GE network interface cards (NICs) are used at both iSCSI initiator and target. All experiments are based on Redhat Linux 8.0 (kernel version 2.4.20). TCP network performance of the system is tested by netperf. Result shows that throughput can reach 939.8Mbps with maximum 70% CPU utilization.

RAM I/O mode or DISK I/O mode is used in iSCSI target. RAM I/O directly maps requests to memory. In DISK I/O test, to diminish the impact from low speed storage device, we use a 2G fibre channel HBA to connect a self-developed high speed fibre channel disk array, which can reach around 110 MB/s for sequential write and 80 MB/s for sequential read.

We use dd command as the application benchmark tool to copy data to/from iSCSI disk. For example, "dd write" is "dd if=/dev/zero of=/dev/sda bs=4k count=500000". It will generate write request to raw device. The size of request is set much bigger than initiator's memory to avoid impact of local cache. In big I/O test, the general dd commands will generate SCSI commands, which request 128KB data to SCSI layer. In small I/O test, the kernel source code is modified to allow dd command to generate 1KB to 32KB requests to SCSI layer. Different queue lengths are tested for the small I/O test.

4. Experimental Results and Analysis

The iSCSI prototype with multiple connections has been tested and the results have been compared with the normal iSCSI performance. Figure 5 shows the throughput of iSCSI with RAM I/O mode .The I/O request size is 128KB. Ethernet frame size is set as 1.5k (small frame) or 9k (jumbo frame) respectively. The multiple connections iSCSI prototype can achieve 70~80MB/s for small frame test and 97~107MB/s for jumbo frame test, which is around 20%~60% improvement compared with normal iSCSI prototype. Further analysis of CPU utilization shows that utilization of initiator's CPU can reach almost 100% with 107MB/s write throughput. That means CPU's power become system's bottleneck in this test condition.

The impact of the frame size on the iSCSI performance is shown in Figure 6. The bigger frame will improve performance mainly because it decrease interrupt times. The impact is very small when the frame size is bigger than 3k. CPU utilization in the initiator is always higher than target. This is because the initiator needs to handle one more data copy in memory than target, when RAM I/O is employed in target.

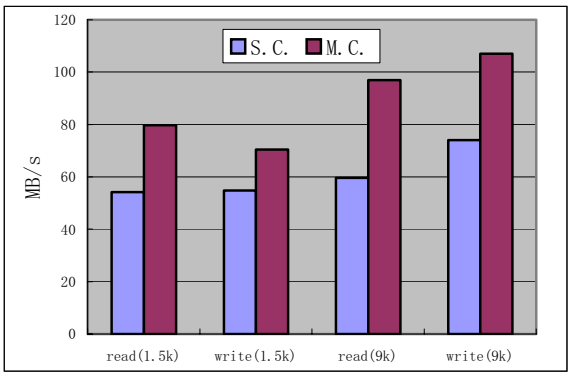


Fig. 5 single /multiple connection of RAM I/O

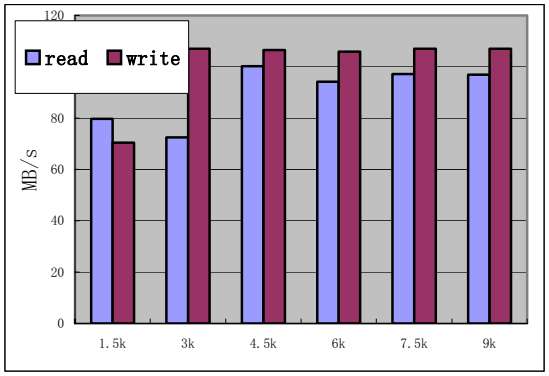


Fig. 6. Effects of Ethernet frame on the throughput

The performance of multiple connection iSCSI for small I/O request is tested. Since the queue depth is a critical parameter that affects the iSCSI performance in small I/O, the effects of the queue length on iSCSI performance is summarized in Figure 7. The testing condition is set as request size with 1KB and frame size with 1.5kB. When queue depth equals to 2, I/O rate is only about 4000 IOPS. When the queue length is bigger than 8, I/O rate can reach around 15000 IOPS. The results show that the I/O rate increases with the queue length until the queue length equals to 8. Further analyzing the captured data by iSCSI analyzer, we find that the maximum effective command queue length is 8 in our prototype and experiments.

Figure 8 shows test result of DISK I/O in the lab and metro network environment. In the lab environment, read performance of iSCSI working in DISK I/O mode is much lower than that in RAM I/O mode. This is because of one more data copy to hard disk in the iSCSI target. For write operation, write cache of raid array card in the iSCSI target makes the iSCSI write performance in DISK I/O mode almost same as that in RAM I/O mode.

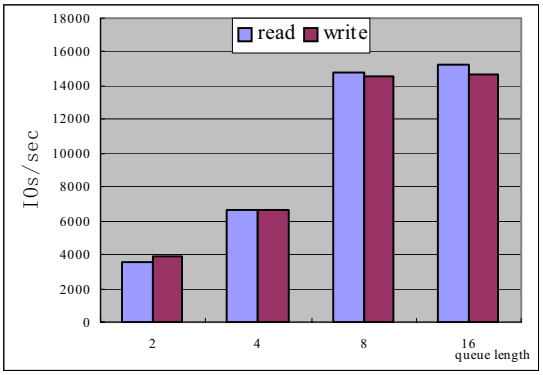


Fig.7 I/O rate for small I/O on different queue length

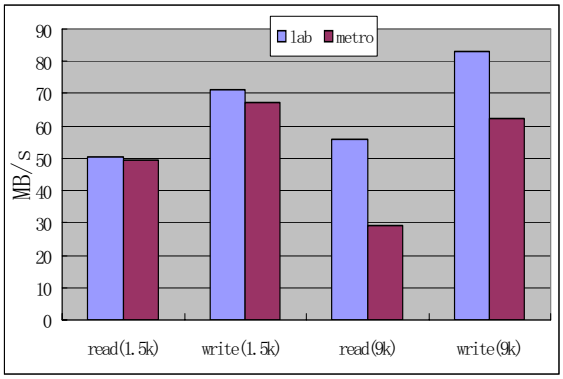


Fig.8 result of lab and metro network of DISK I/O

It is commonly supposed that network latency has a big impact on the iSCSI performance in a metro network. Network latency is estimated by the ping command. The roundtrip time is around 0.6ms in metro network which is around 20 times of that in lab network. But the throughput of our new iSCSI prototype can still reach 67MB/s with small frame in the experiment, which is only 2%~5% less than that of lab. This is due to the iSCSI queue architecture and multiple connections design. It seems that the metro network can't support jumbo frame well because it makes performance much worse than small frames.

5. Conclusion

The paper presents a new iSCSI design with multiple TCP/IP connections. The prototype has been developed and tested in both lab and metro network environment. Results show that the new iSCSI prototype can achieve 20%~60% performance improvement compared with normal iSCSI architecture with single TCP/IP connection. The new iSCSI architecture has been proved in the metro network environment. Future work will focus on the iSCSI testing and application in real network environment.

Reference

[1] Rodney Van Meter, Gregory G, Finn, Steve Hotz, "VISA: Netstation's Virtual Internet SCSI Adapter",
Proceedings of the eighth international conference on Architectural support for programming languages and operating systems, 1998

[2] Prasenjit Sarkar , Kaladhar Voruganti , "IP Storage: The Challenge Ahead"
Proc of 10th conference on Mass Storage Systems and Technologies,April-2002.

[3] Kalman Z.Meth, "iSCSI Initiator Design and Implementation Experience",
Proc of 10th conference on Mass Storage Systems and Technologies, April-2002.

[4] Wee Tech Ng, Bruce Hillyer, Elizabeth Shriver, Eran Gabber, Banu Özden, "Obtaining High Performance for Storage Outsourcing"
Proceedings of the Conference on File and Storage Technologies, 2002

[5] Xubin He, Qin Yang, Ming Zhang, " A Caching Strategy to improve iSCSI Performance",
Proc of the 27th Annual IEEE conference on Local Computer Networks (LCN'02), 2002.

[6] Alacritech, White paper, "Delivering High Performance Storage Networking",
http://www.alacritech.com/html/storagewhitepaper.html.

[7] Adaptec, White paper, "Building SANs with iSCSI, Ethernet and Adaptec",
http://www.graphics.adaptec.com/pdfs/buildingsanwithiscsi-21.pdf

Quanta Data Storage: A New Storage Paradigm

Prabhanjan C. Gurumohan
Arizona State University
gpj@asu.edu

Sai S. B. Narasimhamurthy
Arizona State University
saib@asu.edu

Joseph Y. Hui
Arizona State University
jhui@asu.edu

Abstract

TCP layer and poor iSCSI implementations have been identified as the main bottlenecks in realizing high iSCSI performance. With the addition of security mechanisms the throughput achieved by the storage system using iSCSI is further reduced. Along with the above mentioned problems, we argue that the excessive processing redundancy introduced by several protocol layers and use of protocols designed for non-storage specific requirements result in poor storage network architectures. In order to overcome these issues, we introduce a new storage paradigm in which data is manipulated, encrypted and stored in fixed block sizes called quanta. Each quanta is manipulated by a single effective cross layer (ECL) that includes security features, iSCSI functionalities, direct data placement techniques and data transport mechanisms. Further, the new architecture emphasizes majority of burden of computation for achieving security on the clients.

Qualitative description of the idea is presented. Performance improvements observed during tests of the idea are presented. Through emulation and analysis we also show that the size of the quanta must be equal to the minimum path MTU for maximum throughput.

1. Introduction

IP has been accepted as a de-facto standard for Internet applications. IP based networks also provide relatively inexpensive and convenient solution [5][13] for transportation of bulk data. Considering these facts, transportation of storage data on the IP network provides a cheap and easy alternative to the SCSI and Fiber Channel based networks. iSCSI [5] is the proposed protocol for storage (block) data transport over IP networks. Most effort has been concentrated on designing the protocol over the existing TCP/IP protocol. Initial versions of the implementation of this protocol are available. Although the idea to develop these new protocols over existing protocols was done to ease the integration of iSCSI, it has resulted in over layering and crowding of protocols stacks.

Stephen et. al [11] have presented the findings of implementing an iSCSI based *target* on a specialized hardware. Their work concentrated on comparing the overall performance of various network configurations using iSCSI protocol. The performance results from their work indicate that the iSCSI protocol can be severely limited by implementation. They suspect this problem due to inefficient handling of underlying network properties and poor iSCSI implementation.

Y.Lu and D.Du [8] have examined different storage protocols (viz, iSCSI, NFS,SMB) and have analyzed their performance. The work shows that iSCSI storage with Gigabit connection could have performance very close to directly attached fiber channel-arbitrated loop storage. From their study they also show that iSCSI based file access performance outperforms the NAS schemes on both Windows and Linux platforms. However, they point out that the advantage of iSCSI reduces as the file size increases. The reasoning for this reduction in performance has not been presented in this work.

Shuang-Yi Tang et. al [10] have also shown that with addition of security features such as IPsec below iSCSI and TCP/IP protocol stack leads to high degradation in throughput. Further the idea of storage security is not well severed by usage of IPsec because it does not encrypt the data that will be stored on the storage device. It only protects from attacks between two communicating points. Hence, using iSCSI over TCP/IP/IPsec leads to high overhead and complex layering structure. Further, use of IPsec leads to encryption and decryption at both sever and client ends.

End system copies and TCP packet reassembly form throughout bottlenecks for many applications including iSCSI [1][2][3]. End system copies can be eliminated by Direct Data Placement (DDP) where the application entities are directly placed into application buffers without system copies [1]. DDP packets have to be modified to suite the existing TCP functions. This is enabled by a Framing protocol for TCP [3]. Additional extensions to the DDP protocol are provided by the RDMA protocol, which operates over the DDP protocol. The RDMA, DDP and the Framing

protocol form the iWARP suite. This is the proposed solution for end system copy issues [12]. The iWARP suite is also applicable for iSCSI [2]. The iWARP suite is proposed to operate between iSCSI and TCP [12].

The introduction of iWARP leads to increased overheads and redundancy. Two simple examples of redundant functions at different layers are as follows. First, the CRCs/checksums are computed at the iSCSI layer, MPA or framing layer, and the TCP layer. Second, sequencing information is repeated in both iSCSI and transport layer.

Motivated to overcome these problems, we propose the use of fixed block size data units called *quanta*. These data units are chosen to be of size 512, 1024, 2048 or 4096 bytes. A quanta is encrypted and formatted according to an Effective Cross Layer (ECL) by the client wishing to write data to a server and is stored *as is* on the storage device. The effective cross layer combines the functionalities of encryption, buffer management for direct data placement, iSCSI formatting and transport functions. The key used for encryption of such a quanta is stored on a separate key server. Any valid client can access the encrypted and preformatted data from the server and decrypt it using keys obtained from the key server. The client does the encryption and decryption, checksum calculation, and any other formatting. This lets the target to be implemented with fewer functionalities of ECL and used for only providing an access point to the storage device.

A facility called *fast buffers* (fbufs) was introduced by Peter Druschel and Larry L. Peterson [14]. It was an operating system facility implemented for I/O buffer management and data transfer across protection domain boundaries on shared memory machines. Although, the main goal of this idea was to provide high bandwidth to user level processes, this was achieved by implementing a new facility in the operating system. This is unlike the ECL and quanta approach that achieves the similar goal by reducing the overheads in the protocol stacks. Further, ECL and quanta approach are specifically designed to improve the performance of the storage networks instead for the general networks.

David D. Clark and David L. Tennenhouse [15] provide guidelines for designing of new generation of protocols. Our protocol design efforts are in line with two important guidelines presented in [15]. They are, the data manipulation costs are more compared to transfer control operations and the application data units are the natural pipelining units.

In section 2, we provide the architectural goals of the cross layer design. Section 3 discusses the encryption mechanism details. We present the ECL details in section 4. Section 5 provides the test results for the emulation of ECL using Hyperscsi. Finally we present the conclusion in section 6.

2. Cross layer architectural goals

CLA (Cross Layer Architecture) enables the combining of the features of the SCSI/iSCSI/RDMA /DDP/Framing/TCP/IPsec into a single layer called the ECL. This obviates layering overheads and enables the ease of operation of Storage Area Networks.

CLA has been designed with the following architectural goals:

1. Minimize data handling and processing by dividing data units into fixed size blocks or quanta.

2. Provide security on the wire and on the storage unit.

3. Provide the features of the iSCSI protocol.

4. Provide application level buffering by incorporating the features of the DDP protocol. (Direct Data Placement)

5. Provide the transport layer mechanisms.

3. Encryption Mechanism

In order to include a security mechanism as a part of a single layer, ECL, a new security mechanism is proposed. The new scheme for storage security mainly deals with reducing the high overheads of authentication and encryption. This is done by avoiding encryption and decryption of data on both client and server sides. The new scheme does both encryption and decryption on the client side. The new scheme includes several techniques, which improves the efficiency of encryption and decryption:

1. The data is stored in the encrypted form on the server:

2. Encryption and Decryption is never performed on the server and hence there is no key management in the server.

3. In addition to encryption, the computation of parities, packetization, and error-corrective information is done at the time the file is stored and never recomputed again.

4. The majority of the computation load for providing security and error correction is performed at the client end.

5. Authentication, key management is decoupled from data security.

These requirements are met by implementing a new security mechanism. The data to be stored is encrypted by the client and preformatted. This data unit is called the Encrypted Data Unit (EDU). The EDU is stored on the servers *as is*. The EDU is transported to be stored on servers using ECL headers. The combination of EDUs and the ECL headers are called quanta. The keys used to encrypt the data are generated using AES encryption algorithm. The keys belonging to a particular file is stored in a file and encrypted. This file is stored in a centralized key server. A valid client can access the file on servers stored in the quanta form. The clients fetch the keys from the key server and perform decryption. Figure 1 shows the network architecture that would be used to implement the security mechanism.

Several parameters that are used for read and write operation, encryption, and decryption are included in the ECL header. This is discussed in the next section.

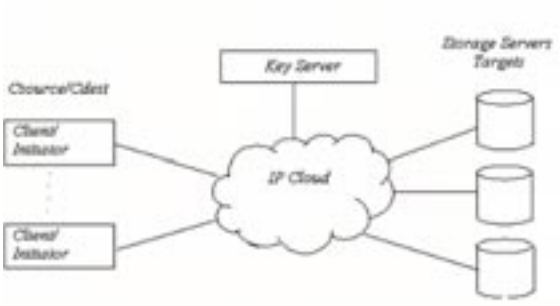


Figure 1. System Setup for data storage centric network

4. Effective Cross Layer

A single layer satisfying the specified objectives is defined as the ECL. Figure 2 shows the iWARP stack for iSCSI along with the security layer, IPsec. Figure 3 shows the corresponding ECL. In the following paragraphs we identify several common features in different layers. The necessary and common functionalities are then retained and incorporated into the ECL and the rest are excluded. These functionalities are designed such that they are scalable over a WAN.

4.1. ISCSI functionalities of the ECL

iSCSI is an adaptation of SCSI for accessing data over the Internet. Hence most of the SCSI functionalities that are part of iSCSI are retained in ECL [5]. Headers and data digests in iSCSI were not inherited from SCSI. These were added for error correction in the iSCSI protocol. Hence header and the data digests are excluded from the ECL. Instead, a single checksum for the entire quanta traversing the channel are utilized.

4.2. Copy avoidance functionalities of the ECL

Most of the iWARP functionalities are retained in the ECL. Messages framed using MPA protocol is obviated by defining a constant MTU in the SAN. Further, the quanta size is always chosen less than or equal to the minimum MTU.

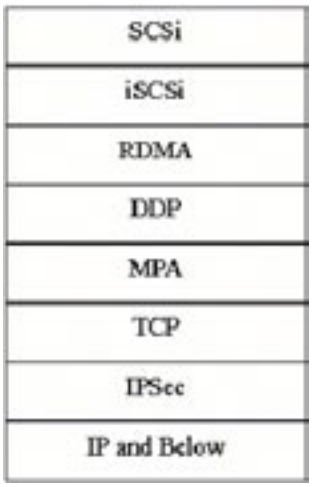


Fig 2. iWARP suite for iSCSI

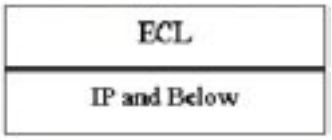


Fig 3. Effective Cross Layer

This obviates the necessity of fragmentation and thus there is no need for markers. The minimum MTU over a path between a source and the

destination can be discovered, before any transmission occurs.

The DDP protocol requires the *tags* or the addresses of buffers at the destination. These tags when available at the source enable *direct data placement*. As a consequence the quanta do not subject themselves to reassembly buffering and kernel copies. Hence they are directly placed at the appropriate SCSI buffers from the NIC.

4.3. Transport functionalities of the ECL

Sequencing [6] information is retained from the iSCSI headers. Thus the sequencing information at the TCP layer can be excluded. Congestion and flow control [6] algorithms are implemented in TCP and they are based on *sliding windows* [6]. The windows are based on sequence numbers. ECL has sequencing information similar to TCP. Hence congestion and flow control mechanisms similar to that of TCP can be used.

Source and destination ports [6] identification is due to a requirement of socket level de-multiplexing. Instead direct data placement provides buffer addressing information that can be used to place the data directly into application buffers. Thus direct data placement alleviates the need for source and destination port information.

The data checksum is computed during the encryption process. Therefore there is a need only for a single quanta header checksum. Message length information that is a part of UDP can be excluded because it is also a part of the iSCSI functionality.

4.4. Security considerations in ECL

The encryption should be done only when a write or an update operation is done. During a read operation, only decryption is required. These operations are indicated in the iSCSI functionality. Hence it is not necessary to add it into the ECL separately. Authentication must be performed before every transaction. Authentication is done by the login mechanism. iSCSI login mechanism that are retained can be employed for this purpose.

The final ECL header structure that combines the features mentioned above is shown in figure 4.

5. Emulation of ECL by HYPERSCSI

In order to evaluate the performance improvements with the use of ECL, tests were conducted using Hyperscsi [7]. Hyperscsi was used because it emulates a simple version of the ECL. In the current form, Hyperscsi is equivalent to SCSI packets placed directly over Ethernet that does not involve copies, reassembly buffering, and functionality redundancies of layers.

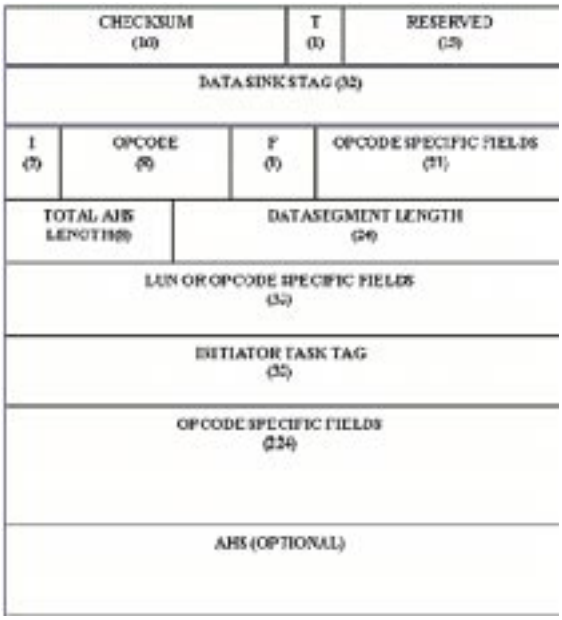


Fig 4. ECL header for WRITE operations

The test setup for ECL throughput characterization consists of two Dell Power Edge-Linux boxes (initiator and the target) with Pentium 866 MHz processors connected end to end through a Gigabit Ethernet link (82546 Dual port GBE). Linux Kernel 2.4.20-18.7 was used in the end systems. Throughput was measured using the bonnie++1.03 tool [16]. Tcpdump and Ethereal [17] was used to see the packet dumps on the end systems. iSCSI Reference 0.18 v10 [18] was used for comparison with iSCSI on TCP. 40 GB QUANTUM hard disks with ULTRA-160 SCSI bus were used at the target. Figure 5 shows the throughput snapshot tests indicating throughput improvements with the ECL.

Several read operations were performed in three different scenarios. First, read operations were performed using only the SCSI protocol. Second was performed using Hyperscsi and the third repeated with UNH iSCSI v10. The results represent an arithmetic mean of 10 trials for a fixed MTU size of 16000. The read throughput achieved using Hyperscsi was close to direct access, whereas, the latest version of the UNH iSCSI code achieves only 219.6 Mb/s as opposed to 358 Mb/s and 363 Mb/s achieved by Hyperscsi and local disk access respectively. The improvement of 63% denotes the extent of betterment in throughput that can be obtained by getting rid of system copies,

reassembly buffering and multiplicity of functionalities in the iWARP stack through the ECL emulated by Hyperscsi.

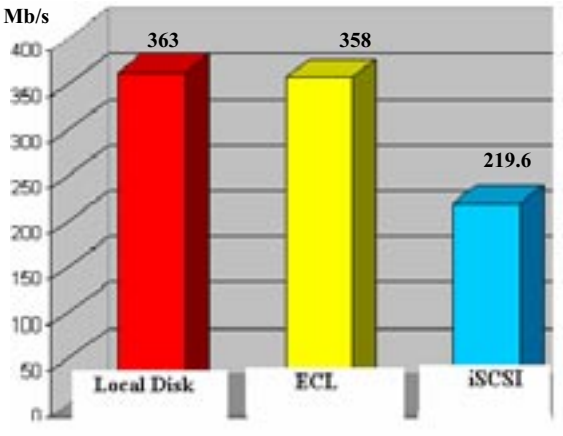


Fig 5. Read performance through the ECL

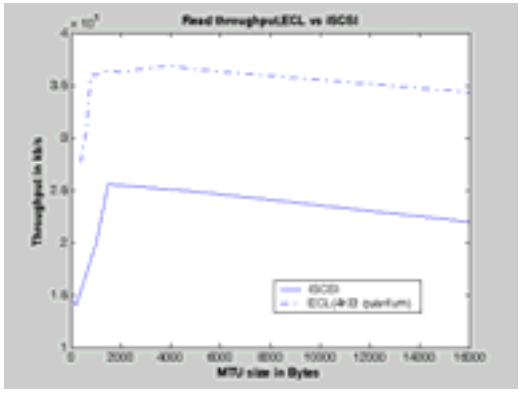


Fig 6. Bulk read throughput results

We next investigate the throughput characteristics of the ECL for reads and writes and justify the quantum data lengths equal to the path MTU.

The ECL needs to accommodate very large windows when it is operating over a network with large round trip times. The required changes are made to the Hyperscsi code to accommodate infinitely large windows. (Hyperscsi at present can accommodate only 32 segments in a window interval). 4K quanta are used for ECL. Fragmentation of quantum is allowed for MTU values less than 4K. For MTU values greater than 4K, each quantum is contained within a single Ethernet packet along with the headers.

Figure 6 compares the iSCSI throughput vs. the ECL throughput for reads for various possible MTU sizes in the Gigabit Ethernet platform. The read throughput shows a constant improvement of about 63% in the favor of ECL.

Figure 7 compares the iSCSI throughput vs. the ECL throughput for writes for various possible MTU sizes in the Gigabit Ethernet platform. The write throughput is less than the read throughput for both the iSCSI and the ECL cases, since writes are more expensive than reads. For large MTU sizes, the ECL throughput approaches the iSCSI throughput since we speculate that non-TCP/IP overheads for large MTU values in an ECL environment approach the TCP/IP overheads for large MTU values in the TCP/IP environment.

Figure 8 shows the read throughput variations for 4K quantum ECL. The throughput peaks at MTU values equal to the quantum-sized blocks. We allow for fragmentation in our tests for MTU sizes less than the quantum sizes. For MTU sizes greater than the quantum sizes, the throughput gradually decays since the per packet non-TCP/IP overheads begin to dominate. This is due to the Ethernet per packet processing overheads. For MTU sizes less than the quantum sizes, the fragmentation overheads exceed the Ethernet per packet overheads. The optimal values are thus reached for MTU sizes almost equal to the quantum sizes. The trend is also seen for writes as shown in figure 9.

An increase in quanta size increases the throughput. The write throughput for 5K path MTU and various quanta sizes are depicted in figure 10.

The tests conducted conclude the throughput characteristics of the ECL and justifies the Quantum size selections equal to MTU sizes.

6. Conclusions

Existing iSCSI storage systems exhibit low performance. Additional protocols have been proposed for improving the performance. We presented the argument that this would lead to further decrease in performance. This is due to excessive processing redundancy and several protocol layers. Further, use of protocols designed for non-storage specific requirements result in poor storage network architectures. In order to solve these problems we proposed data handling in the form of fixed data units called quanta. A new Effective Cross Layer was proposed that combines the necessary features of security, iSCSI, direct data placement, and TCP. A new security mechanism is proposed that emphasizes burden of computation on clients. Throughput improvements over the existing iSCSI are indicated by using the Hyperscsi protocol for emulation. The characteristics of the ECL throughput are noted.

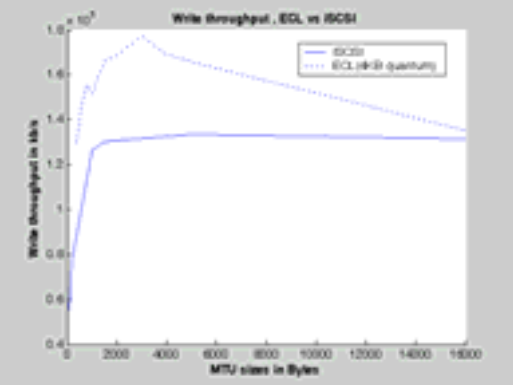


Fig 7. Bulk write throughput results

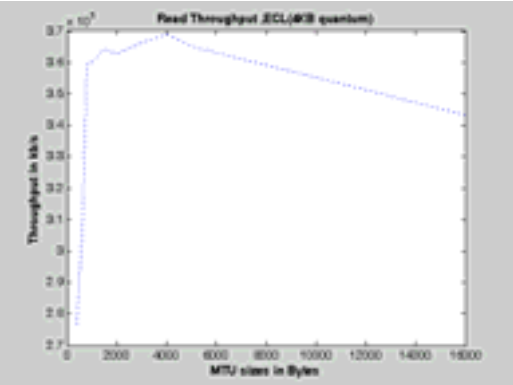


Fig 8. ECL bulk read throughput results

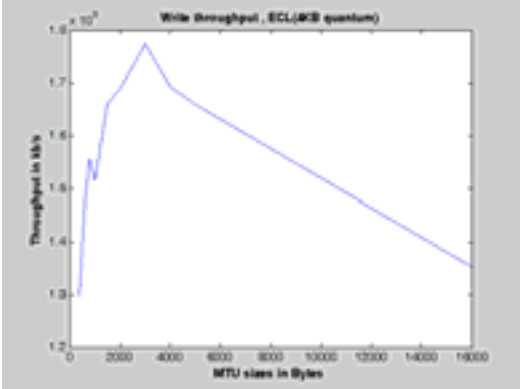


Fig 9. ECL Bulk write throughput results

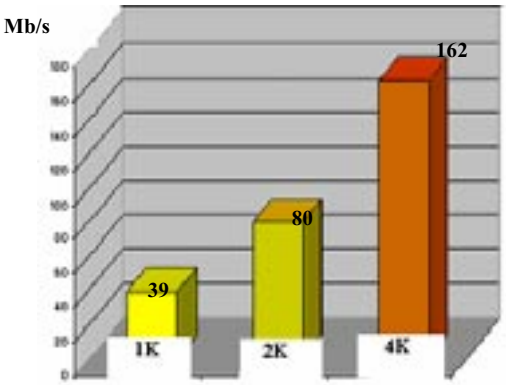


Fig 10. Write Throughput for 1K, 2K and 4K-quantum sizes- in Mb/s for 5K path MTU

References

[1] Hemal Shah, James Pinkerton, Renato Recio, and Paul Culley, *Direct Data Placement over Reliable Transports*, IETF internet draft, draft-shah-iwarpddp-00.txt (Work in progress), October 2002

[2] R.Recio, P.Culley, D.Garcia, and J. Hilland, *RDMA Protocol Specification*, IETF internet draft, draft-recio-iwarp-rdmap-00.txt, October 2002

[3] P.Culley, U.Elzur, R.Recio, and S.Bailey et. al, *Marker PDU Aligned Framing for TCP specification*, IETF internet draft, draft-culley-iwarp-mpa-03.txt, June 2003

[4] S Kent, *IP Encapsulating Security Payload (ESP),* IETF internet draft, draft-ietf-ipsec-esp-v3-06.txt, July 2003

[5] Julian Satran, Costa Sapuntzakis, Mallikarjun Chandalapaka and Efri Zeidner, *iSCSI*, IETF internet draft, draft-ietf-ips-iSCSI-20.txt

[6] R Stevens, *TCP/IP illustrated, Volume 2*, Addison-Wesley, November 2001

[7] **http://www.nst.dsi.a-star.edu.sg/mcsa/**

[8] Yingping Lu and David H.C.Du, *Performance Study of iSCSI-Based Storage Subsystems*, IEEE communications magazine, August 2003

[9] Yongdae Kim, Fabio Maino, Maithili Narasimhma, and Gene Tsudik, *Secure Group Key Management for Storage Area Networks*, IEEE communications magazine, August 2003

[10] Shuang-Yi Tang, Ying-Ping Lu, and David H.C. Du, *Performance Study of Software-Based iSCSI Security*, Proceedings of the First International IEEE Security in Storage Workshop, SISW'02, 2003

[11] StephenAiken, Dirk Grunwald, Andrew R.Pleszkun, and Jesse Willeke, *Performance Analysis of the iSCSI protocol*, IEEE MSST, 2003

[12] **http://www.rdmaconsortium.org**

[13] Rodney Van Meter, Gregory G. Finn, and Steve Hotz, *VISA: Netstation' s Virtual Internet SCSI Adapter*, Proceedings of the eighth international conference on Architectural Support for Programming

Languages and Operating Systems**,** Pages 71 - 80 , October 1998

[14] Peter Druschel and Larry L. Peterson, *A High-Bandwidth Cross-Domain Transfer Facility,* Proceedings of the fourteenth ACM symposium on Operating Systems Principles**,** volume 27 issue 5, December 1993

[15] David D. Clark and David L. Tennenhouse, *Architectural Considerations for a New Generation of Protocols*, ACM SIGCOMM Computer Communication Review, Proceedings of the ACM symposium on Communications architectures & protocols, Volume 20 Issue 4 , August 1990

[16] **http://aixpdslib.seas.ucla.edu/packages/bonnie++.html**

[17] **http://www.ethereal.com/**

[18] **http://sourceforge.net/projects/unh-iscsi/**

Rebuild Strategies for Redundant Disk Arrays

Gang Fu, Alexander Thomasian*, Chunqi Han, and Spencer Ng†
Computer Science Department
New Jersey Institute of Technology -NJIT
Newark, NJ 07102, USA

Abstract

RAID5 performance is critical while rebuild is in progress, since in addition to the increased load to recreate lost data on demand, there is interference caused by rebuild requests. We report on simulation results, which show that processing user requests at a higher, rather than the same priority as rebuild requests, results in a lower response time for user requests, as well as reduced rebuild time. Several other parameters related to rebuild processing are also explored.

1 Introduction

RAID5 with rotated parity is a popular design, which tolerates single disk failure and balances disk loads via *striping*. Striping allocates successive segments of files called *stripe units* on the $N-1$ disks in an array of N disks, with one stripe unit dedicated to parity, so that a capacity equal to the capacity of one disk is dedicated to parity. In this case the *parity group size* G is equal to N. The parity blocks are kept up-to-date as data is updated and this is especially costly when small randomly placed data blocks are updated, hence the *small write penalty*.

If a single disk fails, a block of data on that disk is recreated by exclusive-ORing (XORing) the corresponding blocks on the $N-1$ surviving disks. Each surviving disks needs to process the load due to fork-join requests to recreate lost data blocks besides its own load, so that the load on surviving disks is increased, e.g., at worst doubled when all requests are reads. *Clustered RAID* solves this problem by selecting a parity group size $G < N$, so that the load increase is proportional to the *declustering ratio:* $(G-1)/(N-1)$ [3]. *Balanced Incomplete Block Designs – BIBD* [1, 4] and *random permutation layout* [2] are two approaches to balance disk loads from the viewpoint of parity updates.

The rebuild process is a systematic reconstruction of the contents of the failed disk, which is started immediately after a disk fails, provided a hot spare is available. Of interest is the time to complete the rebuild $T_{rebuild}(u)$ and the response time of user requests versus time: $R(t), 0 < t < T_{rebuild}(u)$. The utilization at all disks, which is equal to u before disk failure occurs, is specified explicitly, since it has a first order effect on rebuild time.

A distinction is made between *stripe-oriented* or *rebuild-unit(RU)-oriented* and *disk-oriented* rebuild in [1]. In the former case the reconstruction proceeds one RU at a time, so that the reconstruction of the next RU is started after the previous one is reconstructed and even written to disk. Disk-oriented rebuild reads RUs from all surviving disks asynchronously, so that the number of RUs read from surviving disks and held in a buffer in the disk array controller can vary. It is shown in [1] that disk-oriented rebuild outperforms stripe-oriented rebuild, therefore the stripe-oriented rebuild policy will not be considered further in this study.

Rebuild requests can be processed at the same priority as user requests, which is the case with the *permanent customer model – PCM* [2], while [1, 6] process rebuild requests when the disk is idle according to the well-known *vacationing server model – VSM* in queueing theory: an idle server (resp. disk) takes successive vacations (resp. reads successive RUs), but returns from vacation (resp. stops reading RUs) when a user request arrives at the disk. In effect rebuild requests are processed at a lower priority than user requests. In PCM a new RU is introduced at the tail of the request queue, as soon as the processing of the previous RU is completed, and hence the name of the model.

The few studies dealing with rebuild processing [1, 2, 6] leave several questions unanswered, such

*The first three authors are partially supported by NSF through Grant 0105485 in Computer Systems Architecture.

†Hitachi Global Storage Technologies, San Jose Research Center, San Jose, CA.

as the relative performance of VSM versus PCM, the effect of disk zoning, etc., but not all issues are addressed here due to space limitations. We extended our RAID5 simulator to simulate rebuild processing. This simulator utilizes a detailed simulator of single disks, which can handle different disk drives whose characteristics are available at: http://www.pdl.cmu.edu/Dixtrac/index.html. The reason for adopting simulation rather than an analytic solution method is because of the approximations required for analysis, which would have required validation by simulation anyway. Simulation results are given in the next section, which is followed by conclusions.

2 Experimental Results

The parameters of the simulation are as follows. We utilize IBM 18ES 9 GByte, 7200 RPM disk drives. We assume an OLTP workload generating requests to small (4KB) randomly placed blocks over the data blocks of the $N = 19$ disks in the array. Track alignment ensures that all 4 KB accesses are carried out efficiently, since they will not span track boundaries. Accesses to randomly placed disk blocks introduce a high overhead, i.e., 11.54 ms per request with FCFS scheduling, less than 1% of which is data transfer time. We assume that the ratio of reads to writes is R:W=1:0, since reads introduce a heavier performance degradation upon disk failure. While we experimented with different disk utilizations, only results for $u = 0.45$, which results in a 90% disk utilization are reported here. We assume zero-latency read and write capability, which has a significant impact on rebuild time.

The parameter space to be investigated includes: (i) VSM versus PCM. (ii) The impact of buffer size per disk B specified as number of tracks. (iii) the size of the RU (rebuild unit) is a multiple of tracks and $RU = 1$ is the default value. (iv) the effect of preempting rebuild requests. (v) The effect of piggybacking, also considered in [1]. (vi) The effect of read-redirection and controlling the fraction of reads redirected. In fact due to its beneficial effect read-redirection is postulated in all other cases reported here [1, 5]. (vii) The effect of the number of disks on rebuild time. (viii) A first order approximation for rebuild time. Due to space limitations item (iv) is not investigated.

2.1 VSM versus PCM

The response time of user requests $R(t)$ and the completion percentage $c(t)$, which is the fraction of tracks already rebuilt, versus time (t) for both VSM and PCM are shown in Figure 1. A disk failed and rebuild was started at time $t = 135$ sec. The graphs shown are averages over ten runs, but even more runs are required to obtain a tighter confidence interval, say at 95% confidence level. The following observations can be made:

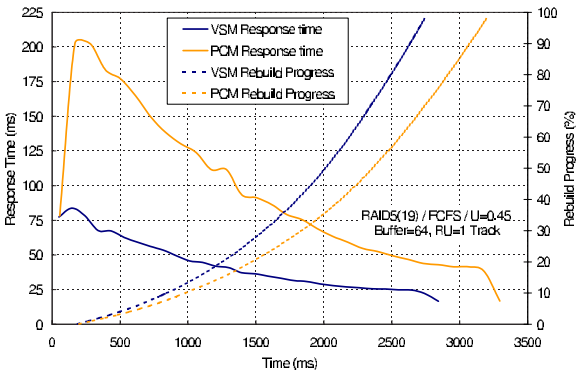


Figure 1: Performance Comparison of VSM and PCM

(i) Disk utilizations are effectivy 100% utilized in both cases, but since RU reads are processed at a lower (nonpreemptive) priority in VSM user requests are only affected by the mean residual service time for RU reads, which is close to half a disk rotation at lower disk utilzations. PCM yields a higher $R(t)$ than VSM since it reads RUs at the same priority as user requests. For each $\bar{n}$ user requests processed (on the average) by a disk, the disk processes $\bar{m}$ consecutive rebuild requests, so that the arrival rate is increased by a factor of $1 + \bar{m}/\bar{n}$. Furthermore, rebuild requests, in spite of zero latency reads, have a longer service time than user requests. (ii) The rebuild time in PCM is higher because during the rebuild period, the disk utilizations due to user requests is approximately the same, but the disk "idleness" is utilized more efficiently by VSM than PCM. The reading of consecutive RUs is started in VSM only when the disk is idle, so that if the reading of an RU is completed before a user request arrives, the reading of the next RU can be carried out without incurring seek time. Uninterrupted processing of rebuild requests is less likely with PCM, since by the time the request to read an RU is being served, it is not likely that the disk queue is empty. This intuition can be ascertained by comparing the mean number of consecutive RU reads in the two cases.

2.2 The impact of buffer size

Figures 2 and 3 show $R(t)$ and $T_{rebuild}$ versus buffer size for VSM and PCM, respectively. We can reduce buffer space requirements by XORing the available blocks right away, but this would introduce con-

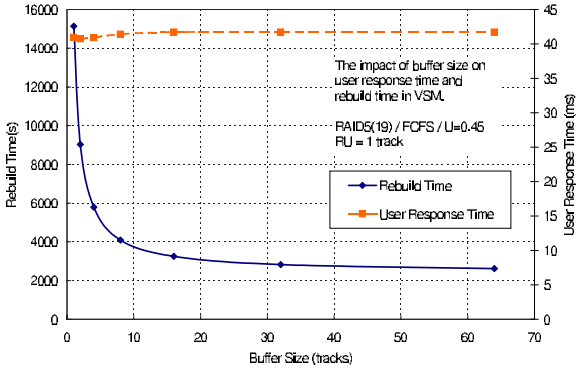


Figure 2: The impact of buffer size in VSM

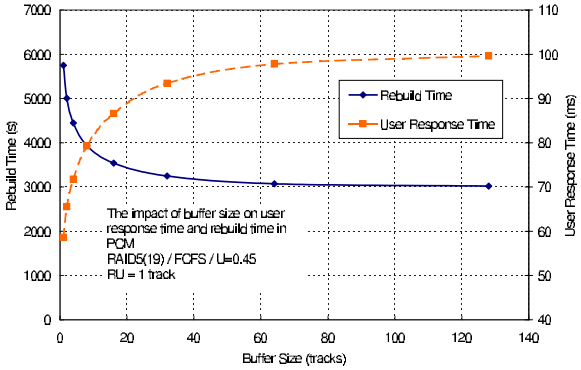


Figure 3: The impact of buffer size in PCM

straints on hardware requirements. Two observations can be made: (i) With disk-oriented rebuild a larger buffer size leads to shorter rebuild times in both VSM and PCM. This is because due to temporary load imbalance, rebuild processing with disk-oriented rebuild is suspended when the buffer is filled. A shared or semi-shared buffer requires further investigation. (ii) The impact of buffer size on $R(t)$ for VSM is very small, but it is significant in PCM. In VSM, the rebuild requests are processed at a lower priority, so the $R(t)$ is only affected by the time to read an RU. In PCM a small buffer size limits the rate of RU reads, i.e., no new RU reads are inserted into the disk queue when the buffer is full, but as B is increased, RU reads are introduced at a rate determined only by u.

2.3 The impact of rebuild unit size

Figures 4 and 5 shows $R(t)$ and $T_{rebuild}$ versus the RU size $= 1, \ldots 16$ tracks. The following observations can be made: (i) The larger RU size leads to higher response times in VSM and PCM. In VSM the mean residual time to read an RU is added to the mean waiting time caused by the contention among user requests [6].(ii) Larger RU sizes lead to shorter rebuild times in VSM and PCM, because the rebuild time per RU is reduced, since the cost of one seek is prorated over the reading of multiple RUs.

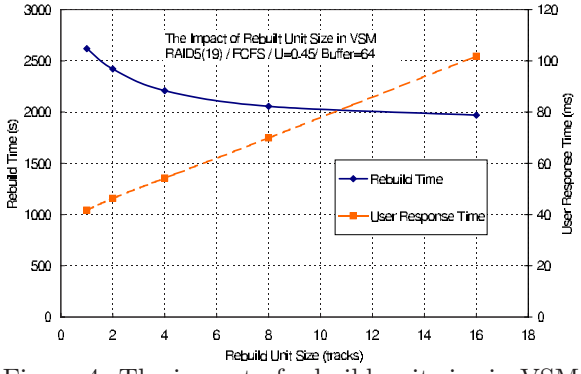


Figure 4: The impact of rebuild unit size in VSM

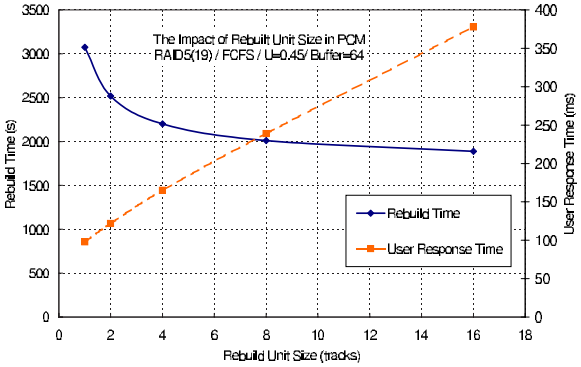


Figure 5: The impact of rebuild unit size in PCM

2.4 The effect of piggybacking

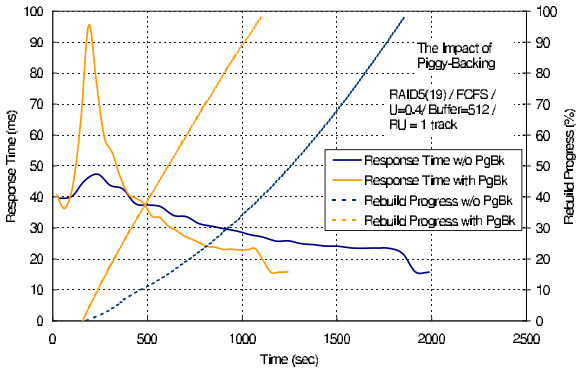


Figure 6: The effect of piggybacking

Figure 6 shows $R(t)$ and $c(t)$ versus t in VSM with and without piggybacking. Piggybacking is done at the RU level, rather than at the level of user accesses, e.g., 4 KB blocks, which has been shown to be ineffectual [1]. In other words, we extend the reconstruction of a single block into a full track. Piggybacking shortens rebuild time, but initially the disk utilization will exceed $2u$ in our case, since track reads have a higher mean service time than the reading of 4 KB blocks, which are carried out on behalf of fork-join requests (this is the reason why $u = 0.4$ in this case). We can control the initial increase in $R(t)$ by controlling the fraction of piggybacked user requests.

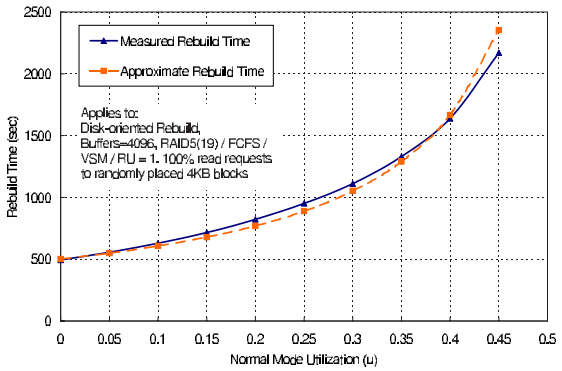


Figure 7: The effect of disk utilization on rebuild time

2.5 The impact of read redirection

Read-redirection shortens the rebuild time (by a factor of three with 19 disks, u=0.45, and B=64) and also reduces $R(t)$ as gradually more tracks from the failed disk are rebuilt. Since we assumed there are no write requests, we get the maximum improvement in performance with read redirection, but there will be less improvement in response time when all requests are updates. In "unclustered" RAID5 the utilization of the spare disk remains below the other disks and there is no need to control the fraction of read requests being redirected, as in [3].

2.6 The effect of the number of the disks

Table 1 gives the rebuild time using VSM versus the number of disks (N) with various number of buffers. The disk utilization is $u = 0.45$. It is observed that even at high disk utilization, rebuild time with VSM is not affected very much by N. An additional observation is that due to the prevailing symmetry, that the load at all disks is balanced, rebuild time is determined by the reading time of any of the disks and that the writing of the spare disk follows shortly thereafter. When the buffer size dedicated to the rebuild process is small, rebuild time is more sensitive to N.

Number of disks		9	19	29	39
Rebuild Time (sec)	B=16384	1604.2	1622.8	1631.9	1632.0
	B=64	2586.3	2618.9	2701.4	2704.5
	B=16	3033.8	3246.0	3434.7	3526.5

Table 1: The effect of number of disks on rebuild time

2.7 The effect of disk utilization on rebuild time

It follows from Figure 7 that rebuild time increases sharply with disk utilization. This is especially so with an infinite source model, where the request arrival rate is not affected by increased disk response time. Rebuild time can be reduced by not processing low priority applications.

We obtain a first order approximation for rebuild time with given disk characteristics by postulating that the rebuild time $T_{rebuild}(u)$ is simply a function of the disk utilization u, so that $T_{rebuild}(u) = T_{rebuild}(0)/(1-\alpha u), \alpha u < 1$, where $T_{rebuild}(0)$ is the time to read an idle disk, and α is a measure of the average increase in disk utilization during rebuild.

With read redirection and R:W=1:0 the utilization of surviving disks initially doubles, but then it drops gradually as disk utilization reaches u, so that $\alpha \approx 1.5$ is an initial guess. On the other hand the rebuild time is slow when rebuild starts, so that more time is spent in the initial stages of rebuild. Curve fitting shows that $\alpha \approx 1.75$ yields an acceptable estimate of rebuild time, but there is a sensitivity to R:W ratio, and other factors, which are being explored.

3 Conclusions

It is interesting to note that VSM outperforms PCM on both counts, user response times and rebuild time. The latter is counterintuitive since PCM rebuilds more aggressively. We are also investigating the effect of preempting rebuild requests, utilizing multiple rebuild regions, and allowing multiple user requests before rebuild processing in VSM is stopped. Finally, we are interested in rebuild processing in RAID6 and EVENODD, especially when operating with two disk failures.

References

[1] M. C. Holland, G. A. Gibson, and D. P. Siewiorek. "Architectures and algorithms for on-line failure recovery in redundant disk arrays", *Distributed and Parallel Databases 11*(3): 295-335 (July 1994).

[2] A. Merchant and P. S. Yu. "Analytic modeling of clustered RAID with mapping based on nearly random permutation", *IEEE Trans. Computers 45*(3): 367-373 (1996).

[3] R. R. Muntz and J. C. S. Lui. "Performance analysis of disk arrays under failure", *Proc. 16th Int'l Conf. VLDB*, 1990, pp. 162–173.

[4] S. W. Ng and R. L. Mattson. "Uniform parity distribution in disk arrays with multiple failures", *IEEE Trans. Computers 43*(4): 501-506 (1994).

[5] A. Thomasian and J. Menon. "Performance analysis of RAID5 disk arrays with a vacationing server model for rebuild mode operation", *Proc. 10th Int'l Conf. Data Eng. - ICDE*, 1994, pp. 111-119.

[6] A. Thomasian and J. Menon. "RAID5 performance with distributed sparing", *IEEE Trans. Parallel and Distributed Systems 8*(6): 640-657 (1997).

Evaluation of Efficient Archival Storage Techniques

Lawrence L. You
University of California, Santa Cruz
Jack Baskin School of Engineering
1156 High Street
Santa Cruz, California 95064
Tel: +1-831-459-4458
you@cs.ucsc.edu

Christos Karamanolis
Storage Systems Group
Hewlett-Packard Labs
1501 Page Mill Road, MS 1134
Palo Alto, CA 94304
Tel: +1-650-857-6956
Fax: +1-650-857-5548
christos@hpl.hp.com

Abstract

The ever-increasing volume of archival data that need to be retained for long periods of time has motivated the design of low-cost, high-efficiency storage systems. Inter-file compression has been proposed as a technique to improve storage efficiency by exploiting the high degree of similarity among archival data. We evaluate the two main inter-file compression techniques, data chunking and delta encoding, and compare them with traditional intra-file compression. We report on experimental results from a range of representative archival data sets.

1 Introduction

Over the last several years, we have witnessed an unprecedented growth of the volume of stored digital data. A recent study estimated the amount of original digital data generated in 2002 alone to be close to 5 exabytes, approximately double the volume of data created in 1999 [4]. An increasing fraction of this corpus is archival data: immutable data retained for long periods of time for legal or archival purposes. Examples of archival data include rich media such as audio, images and video, documents, email, and instant messages.

The high rate of archival data generation has motivated a number of research projects to look into ways of improving the space efficiency of disk-based archival storage systems. Researchers have observed that they can take advantage of content overlapping, which is common in archival data, to improve storage efficiency [9, 3]. There are two main techniques proposed for this purpose. The first technique divides each data object into a number of non-overlapping *chunks* and stores only *unique* chunks in the archival storage system. Chunks may be of fixed or variable size. The second technique is based on resemblance detection between data objects and uses *delta encoding* to store only deltas instead of entire data objects.

These two approaches have been developed and used in very different contexts, with different goals and data sets. For example, variable-size chunking was proposed for improving the bandwidth consumption of network file systems [7]. Delta encoding [1] has been used for data compression in HTTP [6] as well as in version-control systems [11]. However, there has been no attempt to compare the two approaches side-by-side, evaluating the storage efficiency they achieve and their applicability to different archival data sets.

The goals of this work can be summarized as follows: (i) evaluate the applicability of the approaches on different data types that exhibit different degrees

of inter-file similarities; (ii) identify the key parameters for each technique and provide rules of thumb for their settings for different data types; (iii) compare inter-file compression techniques with traditional lossless intra-file techniques and explore the potential benefits of hybrid approaches. Further, we provide a performance analysis of the different approaches, and discuss system design and engineering considerations.

2 Overview

Archival data, by its nature, often exhibits strong inter-file resemblance. This paper examines techniques that take advantage of such inter-file resemblance to avoid storing redundant data and thus improve storage efficiency. Such techniques may be combined, if necessary, with lossless intra-file compression, such as sliding-window compression techniques (e.g. *zip* variants).

Several systems that exploit data redundancy at different levels of granularity have been developed in order to improve storage efficiency. One class of systems detects redundant chunks of data at granularities that range from entire file (EMC's Centera) down to individual fixed-size disk blocks (Venti [9]) and variable-size data chunks (LBFS [7]). We focus on the use of variable-sized chunks, which have been reported to exhibit better efficiency over the special case of fixed-size blocks [8]. Typically, such techniques are used in content-addressable storage (CAS) infrastructures. The second class of systems detects and stores only differences (deltas) between similar files, at the granularity of bytes [3].

2.1 Chunking

Data chunking involves two problems. First, a data stream, such as a file, needs to be divided into chunks in a deterministic way. We consider the general case of variable-sized chunks, which works for any type of data, including binary formats. Chunk boundaries are defined by calculating some feature (a digital signature) over a sliding window of fixed size. In our prototype, we use Rabin fingerprints [10], for their computational efficiency in the above scenario. Boundaries are set where the value of the feature meets certain criteria, such as when the value, modulo some specified integer divisor, is zero; the divisor affects the average chunk size. Such deterministic algorithms do not require any knowledge of other files in the system. Moreover, chunking can be performed in a decentralized fashion, even on the clients of the system.

The second problem is to uniquely identify chunks. An algorithm is required that computes a digest over a variable-length block of data. Currently in the prototype, we reuse the Rabin fingerprinting code for deriving chunk identifiers. In practice and for very large data sets, one would need an algorithm that guarantees low probability for collisions, such as MD5 and SHA variants. Exactly because chunks are content-addressable, chunking is suitable for CAS systems. Only unique chunks are stored for any file. The original files can be reconstructed from their constituent chunks. To do that, the system needs to maintain metadata that maps file identifiers to a list of chunk identifiers. Any evaluation of storage efficiency must take into account the overhead due to the metadata. Figure 1 illustrates the main parameters of a chunking technique.

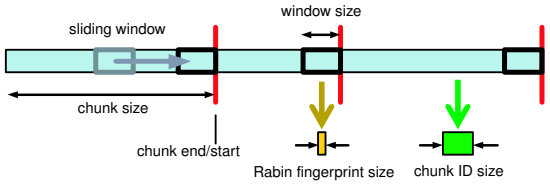


Figure 1: Chunking parameters

We developed a prototype program, named *chc*, to evaluate the efficiency and performance characteristics of chunking without having to build a complete storage system. The input to *chc* is an archive (*tar* file) of a number of files that form the target data set; *chc* produces an output archive that includes unique chunks derived from the original files. Optionally, we can compress the individual chunks in the archive using the *zlib* compression library. *chc* captures a list of chunk identifiers for each file, as well as the identifier and size for each chunk. This metadata is stored also in the output archive and provides an estimate of storage overhead due to chunking. In addition, *chc* can reconstruct the original data set from the final chunk archive.

2.2 Delta encoding

Delta compression is used to compute a delta encoding between a new file and a reference file already stored in the system. When resemblance is above some threshold, a delta is calculated and only that is stored in the system. There are three key problems that need to be addressed in delta encoding.

First, resemblance has to be detected in a content-independent and efficient way. We use the shingling technique proposed by the DERD project [3]. It calculates Rabin fingerprints using a sliding window along an entire file (the window size is a configurable

parameter). The number of fingerprints produced is proportional to the file size. A deterministic *feature selection* algorithm selects a subset of those fingerprints, called a *sketch*, which is retained and later used to compute an estimate of the *resemblance* between two files by comparing two sketches using the *approximate min-wise independent permutations* [2]. This estimate computes similarity between two files by counting the number of matching pairs of features between two sketches. It has been shown that even small sketches, e.g. sets of 20 features, capture sufficient degrees of resemblance.

Thus, when new data needs to be stored, one has to find an appropriate reference file in the system: a file exhibiting a high degree of resemblance with the new data. In general, this is a computationally intensive task (especially given the expected size of archival data repositories). In the prototype, we use an exhaustive search over all stored files. We are currently investigating the use of hierarchical clustering of sketches to reduce the search.

The third problem is to calculate the delta encoding once a reference file has been found. Delta compression is a well-explored area, and in the prototype we used the *xdelta* tool [5], which computes the output using the *zlib* (*gzip*) library. Pointers to reference files are stored with every delta. These identifiers (e.g. SHA digests), along with sketch data, contribute to accounted storage overhead. Our prototype consists of three programs, one for each of the three above problems: feature extraction, resemblance detection, and delta generation.

3 Evaluation

We explain the experimental methodology we used to measure the efficiency, describe the data sets, and analyze the storage efficiency and differences in performance of the two archival storage techniques.

As noted above, storage efficiency is the determining factor for the applicability of an inter-file compression approach. We report on the storage space required as a percentage of the original, uncompressed data set size. For example, stored data that is 20% the size of the original represents an efficiency ratio of 5:1. The functionality and performance of each approach depends on the settings of a number of parameters. As expected, experimental results indicate that no single parameter setting provides optimal results for all data sets. Thus, we first report on parameter tuning for each approach and different data sets. Then, using optimal parameters for each data set, we compare the overall storage efficiency achieved by each approach. The required storage includes the overhead due to the metadata that needs to be stored. Last, we discuss the performance cost and the design issues of applying the two techniques to an archival storage system.

3.1 Data Sets

We chose a range of data sets that we believe to be representative of archival data. Email messages often contain headers (and sometimes attachments) that show great resemblance. Source code and web content are typically versioned. Non-textual content such as presentations and imagery are often similar and require lots of storage space. Finally, computer-generated data such as logs are generated in high volumes and can contain repeated content such as field descriptors. The following is the list of data sets we use.

- HP Support Unix logs (two sets of different total volume)
- Linux kernel 2.2 source code (four versions)
- Email (single user)
- Mailing list archive (BLU)
- HP ITRC Support web site
- Microsoft PowerPoint presentations
- Digital raster graphics (California DRG 37122 7.5 minute untrimmed TIFF)

3.2 Parameter Tuning

In the case of chunking, the *expected chunk size* is a key configuration parameter. It is implicitly set by setting the fingerprint divisor as well as the minimum and maximum allowed chunk size. In general, the smaller it is, the higher the probability of detecting common chunks among files. For data with very high inter-file similarity (such as log files), small chunk sizes result in greater storage efficiency. However, for most data this is not the case, because smaller chunks also mean higher metadata overhead. Often, because of this overhead the storage space required may be greater than the size of the original corpus. As Figure 2 shows, the optimal expected chunk size depends on the type of data; using 128-bit identifiers, the best efficiencies range from 256 to 512 bytes.

The main configurable parameter in the case of delta encoding is the size of the sketches—i.e. the number of features used for resemblance detection. Our experimental results are consistent with what was reported by Douglis et al.: a sketch size of 20 to 30 features is sufficient to capture resemblance among files. Another parameter is the *resemblance threshold*, the

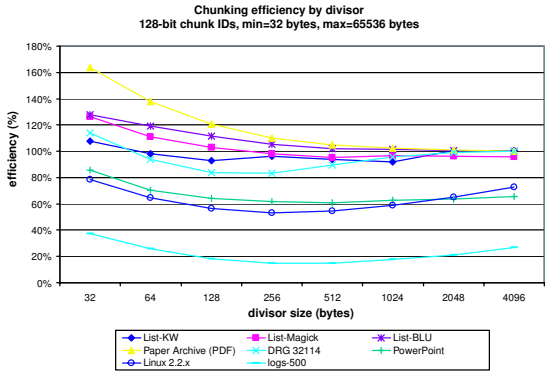


Figure 2: Chunking efficiency by divisor size

number of features that must correspond between two files to consider that sufficient resemblance exists to justify calculating the delta instead of storing the entire new file. For the evaluation of delta encoding, we traverse the target data set one file at a time in a random order. For a file, a delta is created against the file with the highest resemblance that is already in the output archive, as long the resemblance is above a threshold of one corresponding feature. Otherwise, the entire file is stored as a new reference file.

3.3 Storage Efficiency

Table 1 shows the achieved storage efficiency by the two approaches. We do that with and without additional *zlib* compression of chunks and deltas respectively. To establish a baseline for each data set, we create a single *tar* file from the data set, and then compress it with an intra-file compression program, *gzip*. As expected (and as shown by the two first rows of the table), inter-file compression improves with larger corpus sizes. This is not the case with *gzip*.

The HP Unix Logs (8,000 files) show very high similarity. Chunk-based compression on this similar data was reduced to 11% of the original data size, and when each chunk is compressed using the *zlib* (similar to *gzip*) compression, it is just 7.7% of the original size. Even more impressive are the reductions in size when using delta compression. When delta compression is used alone, the data set is reduced to 4% of the original size, but when combined with *zlib* compression, the compressed data is less than 1% of the original size.

Textual content, such as web pages, can be highly similar. However, in the case of the HP ITRC content, *gzip* compression is more efficient than chunking or delta. More surprisingly, *gzip* is better even when we do additional compression of chunks and deltas. The reason is that *gzip*'s dictionary is more efficient across entire files than within the smaller individual chunks, and chunk IDs appear as random (essentially non-compressible) data. But in the context of an archival storage system, *gzip*'s advantage is not likely to be as effective in practice; this is discussed below.

Non-textual data, such as the PowerPoint files with chunking and delta (especially with *gzip*) achieve better efficiency than *gzip* alone. However, the achieved compression rates are less impressive than those for the log data. For raster graphics, delta encoding with *gzip* achieves modest improvement over *gzip* alone.

A single user's email directory and a mailing list archive show little improvement when using delta. Chunking is less effective than *gzip*, although we would expect it to reduce redundancy found across multiple users' data.

In most cases, inter-file compression outperforms intra-file compression, especially when individual chunks and deltas are internally compressed. Chunking achieves impressive efficiency for large volumes of very similar data. On the other hand, delta encoding seems better for less similar data. We believe that this is due to the lower storage overhead required for delta metadata. Typical sketch sizes of 80 to 120 bytes (20 to 30 features $\times$ 4 bytes) for a file of any size are significantly smaller than the overhead of chunk-based storage, which is linear with the size of the file.

Although compressing a set of files into a single *gzip* file to establish a baseline measurement helps illustrate how much redundancy might exist within a data set, it is not likely that an archival storage system would reach those levels of efficiency for several reasons. Most important is that files would be added to an archival system over time and files would be retrieved individually. If a new file were added to the archival store, it would not be stored as efficiently unless the file could be incorporated into an existing compressed file collection, i.e. the new file would need to be added to an existing *tar*/*gzip* file. Likewise, retrieving a file would require first extracting it from a compressed collection and this would require additional time and resources over a chunk or delta-based file retrieval method.

Our experiments measured the size of an entire corpus, in the form of a *tar* file after it has been compressed with *gzip*. Had we compressed each file with

Data Set	Size	# Files	tar + gzip	Chunk	Chunk + zlib	Delta	Delta + zlib
HP Unix Logs	824 MB	500	15%	13%	5.0%	3.0%	1.0%
HP Unix Logs	13,664 MB	8,000	14%	11%	7.7%	4.0%	0.94%
Linux 2.2 source (4 vers.)	255 MB	20,400	23%	57%	22%	44%	24%
Email (single user)	549 MB	544	52%	98%	62%	84%	50%
Mailing List (BLU)	45 MB	46	22%	98%	53%	67%	21%
HP ITRC Web Pages	71 MB	4,751	16%	86%	33%	50%	26%
PowerPoint	14 MB	19	67%	55%	46%	38%	31%
Digital raster graphics	430 MB	83	42%	102%	55%	99%	42%

Table 1: Storage efficiency comparison (64-bit chunk IDs)

gzip first and then computed the aggregate size of all compressed files, the sizes for *gzip*-compressed files would have been much larger. For example, in the case of the HP ITRC web pages, *gzip* efficiency would have been 30% of the original size, much larger than the 16% shown in table 1, and larger than the 26% that can be achieved by using delta compression with *zlib*. When delta compression (or to a lesser extent, chunking) is applied across files first and then an intra-file compression method second, it is more effective than compressing large collections of data because additional redundancy can be eliminated.

3.4 Performance

In practice, space efficiency is not the only factor used to choose a compression technique; we briefly discuss some important systems issues such as computation and I/O performance.

The chunking approach requires less computation than delta encoding. It requires two hashing operations per byte in the input file: one fingerprint calculation and one digest calculation. In contrast, delta encoding requires $s+1$ fingerprint calculations per byte, where s is the sketch size. It also requires calculating the deltas, even though this can be performed efficiently, in linear time with respect to the size of the inputs. Additional issues with delta encoding include efficient file reconstruction and resemblance detection in large repositories.

The two techniques exhibit different I/O patterns. Chunks can be stored on the basis of their identifiers using a (potentially distributed) hash table. There is no need for maintaining placement metadata and hashing may work well in distributed environments. However, reconstructing files may involve random I/O. In contrast, delta-encoded objects are whole reference files or smaller delta files, which can be stored and accessed efficiently in a sequential manner. But, placement in a distributed infrastructure is more involved.

4 Conclusions

Inter-file compression is emerging as a technique to improve space efficiency in archival storage systems. This paper provides the first direct comparison of the two main techniques proposed in the literature, namely chunking and delta encoding, and compares them against traditional intra-file compression. In general, both chunking and delta encoding outperform *gzip*, especially when they are combined with compression of individual chunks and deltas. Chunking is computationally cheap and can be easily used in distributed systems. It works well for data with very high similarity. Thus, it is applicable to applications where there are multiple versions of the same data, such as version control systems, and log files. On the other hand, delta encoding is more computationally expensive, but more efficient with less similar data and thus, it is potentially applicable to a wider range of data sets.

Acknowledgments

Lawrence You was supported by a grant from Hewlett-Packard Laboratories (via CITRIS), Microsoft Research, and supported in part by National Science Foundation Grant CCR-0310888. We thank Kave Eshghi and George Forman of Hewlett-Packard Laboratories for their help and insight into the behavior of file chunking. We are also grateful to members of the Storage Systems Research Center at the University of California, Santa Cruz for their help preparing this paper.

References

[1] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer. Compactly encoding unstructured inputs with differential compression. *Journal of the ACM*, 49(3):318–367, May 2002.

[2] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and Systems Sciences*, 60(3):630–659, 2000.

[3] F. Douglis and A. Iyengar. Application-specific delta-encoding via resemblance detection. In *Proceedings of the 2003 USENIX Annual Technical Conference*, San Antonio, Texas, June 2003.

[4] P. Lyman, H. R. Varian, K. Searingen, P. Charles, N. Good, L. L. Jordan, and J. Pal. How much information? 2003. `http://www.sims.berkeley.edu/research/projects/how-much-info-2003/`, Oct. 2003.

[5] J. P. MacDonald. File system support for delta compression. Master's thesis, University of California at Berkeley, 2000.

[6] J. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '97)*, Sept. 1997.

[7] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 174–187, Lake Louise, Alberta, Canada, Oct. 2001.

[8] C. Policroniades and I. Pratt. Feasibility of data compression by eliminating repeated data in practical file systems. First Year Report.

[9] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In D. D. E. Long, editor, *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 89–101, Monterey, California, USA, 2002. USENIX.

[10] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[11] W. F. Tichy. RCS—a system for version control. *Software—Practice and Experience*, 15(7):637–654, July 1985.

An Efficient Data Sharing Scheme for iSCSI-Based File Systems *

Dingshan He and David H.C. Du
Department of Computer Science and Engineering
DTC Intelligent Storage Consortium
University of Minnesota
{*he,du*}*@cs.umn.edu*

Abstract

iSCSI is an emerging transport protocol for transmitting SCSI storage I/O commands and data blocks over TCP/IP networks. It allows storage devices to be shared by multiple network hosts. A fundamental problem is how to enable consistent and efficient data sharing in iSCSI-based environments. In this paper, we propose a suite of data sharing schemes for iSCSI-based file systems and use ext2 as an example for implementation. Finally, we use simulations to verify the correctness of our designs and to study the performances.

1. Introduction

iSCSI [2] combines two popular and mature protocols in the data storage area and the network communication area - SCSI and TCP/IP. Small Computer System Interface (SCSI) enables host computer systems to perform I/O operations of data blocks with peripheral devices. iSCSI extends the connection of SCSI from traditional parallel cables, which could only stretch to several meters, to ubiquitous TCP/IP networks. iSCSI encapsulates and reliably delivers SCSI Protocol Data Units (PDUs) over TCP/IP networks.

iSCSI is expected to expand the coverage of System Area Networks (SAN). One major feature of SAN is to provide a shared pool of storage resources for multiple accessing hosts. As storage devices are not directly attached to any hosts, they can be easily shared. However, data sharing is going beyond sharing storage devices. With data sharing, a single piece of data could actually be shared by multiple clients. The integrity of data content has to be enforced. Therefore, concurrency control mechanism is necessary when multiple hosts could read/write a single piece of data concurrently.

*This project is partially supported by members of DTC Intelligent Storage Consortium (DISC) at UMN, and gifts from Intel and Cisco.

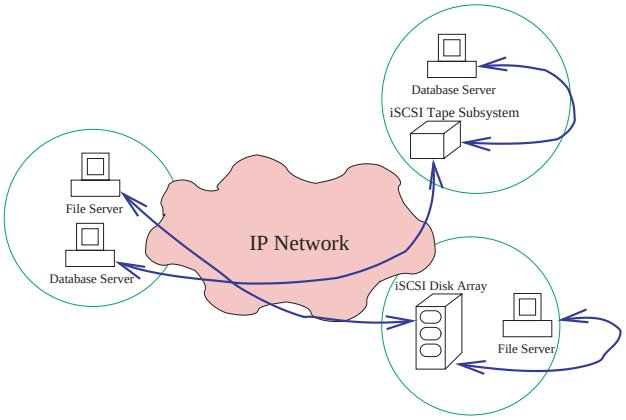


Figure 1. iSCSI network architecture scenario

When using iSCSI-based storage subsystems, application servers (as initiators) and their shared storage subsystems (as targets) could be connected by TCP/IP networks over long distances as depicted in Figure 1. Applications like file systems on the application servers are not aware of the fact that iSCSI-based storage subsystems are accessed and these storage subsystems are potentially shared with other application servers. Remote iSCSI devices are mounted at mounting points of application servers' local file systems and corresponding file system modules are loaded to manage data blocks. The file systems for mounted iSCSI devices are not adapted for iSCSI at all. Although these mounted file system modules in IP hosts are able to use their existing file system locks locally, multiple IP hosts still can not guarantee consistent data sharing. Therefore, our design is to coordinate the accesses of multiple iSCSI initiators.

Since iSCSI-based architectures could be deployed over WANs, latency introduced by large physical distance will be a severe restriction on performance. To improve performance, caching at application servers is necessary. In addition to hiding physical latency, caching at application

servers can also reduce load at iSCSI targets. However, it incurs the problem of cache consistency, which is the consistency between cached data on initiators (application servers) and data on targets (iSCSI-based storage subsystems). In our design, we will enforce strong consistency on cached data by using a callback cache similar to the Coda file system [6]. An iSCSI target keeps track of cached physical blocks on all connected iSCSI initiators. The iSCSI target forces the iSCSI initiators to discard their stale copy, when it is going to modify a physical block.

Metadata are also concurrently accessed, and cached by multiple initiators. However, a general solution for both metadata and normal file data would be inefficient since metadata and normal file data are different in terms of access patterns. We design different mechanisms for them separately as discussed in Section 3.1 and Section 3.2.

Finally, most existing SAN file systems, such as Lustre, only provide UNIX file sharing semantics. However, the UNIX file sharing semantics is not suitable for transactional execution of a sequence of operations. In the transaction file sharing semantics, there should be a consistent view of any involved data throughout the execution of a transaction. Deadlocks are possible, so detection and resolution of deadlocks are considered. In addition, rollback capability is required in case a transaction is unable to complete due to deadlocks.

The rest of this paper is organized as following. In Section 2, we will summarize related work. In Section 3, we are going to give an overview of our design and implementation. In Section 4, we present the simulation results over ns-2 simulator to study the performance of our design.

2. Related Work

The performance of iSCSI-based storage subsystems is studied by Lu and Du in [3]. It is shown that iSCSI storage with Gigabit connection could have performance very close to directly attached FC-AL storage, and iSCSI storage in campus network can achieve reasonable performance restricted by available bandwidth. Tang et al [5] have study performance of software-based iSCSI security using IPSec and SSL. In application side, researchers start to consider using iSCSI to implement hierarchical web proxy server and remote mirroring and backup.

Several distributed or clustered file systems have designed different data sharing schemes. GFS uses Device Lock mechanisms, which has been included in SCSI 3 specification as Dlock command. The IBM Distributed Lock Manager (DLM) is an implementation of the classic VAX Cluster locking semantics. Another simpler implementation is the DLM in Lustre with introduction of object concept. Our work is mainly different from these designs in network environment and locking granularity. We explicitly take possible long network latency into design consideration. Our locking granularity is at physical block level.

Table 1. Compatibility of metadata locks

	M_S	M_X
M_S		+
M_X	+	+

3. Design Overview

Our proposed scheme for data sharing in iSCSI-based file systems consists of two major parts. The first part is a concurrency control mechanism to coordinate multiple concurrent accesses for shared data. The second part is a cache consistency control mechanism. We assume that no single operation will involve data from more than one iSCSI target/LUN.

Roselli et al [1] found that the percentage of metadata reads is much larger than metadata writes. In order to take advantage of this fact, our design allows iSCSI initiators to cache shared locks (referred to as semi-preemptible shared locks [4]) on metadata objects.

On the other hand, normal data are organized into files. The access patterns for files are application dependent. Therefore, in addition to integrity of shared data, we are trying to design a general locking scheme to achieve high concurrency and to reduce maintenance costs of locks. Unfortunately, these two goals are conflicting with each other. Fine granularity of locks are preferred to maximize concurrency, meanwhile coarse granularity reduces number of locking requests and memory space used to maintain locks. Our design is trying to balance between these two conflicting goals. The detail of our hierarchical locking scheme will be discussed later in Section 3.2.

3.1. Locking scheme for metadata

The locks for metadata are applied on metadata objects. We define following 5 kinds of metadata objects: 1) directory files, 2) normal file inodes, 3) super block, 4) inode bitmap blocks, and 5) data-block bitmap blocks.

For each metadata object, there are two possible kinds of locks: M_S and M_X. M_S lock gives shared access to the requested metadata object. M_X lock gives exclusive access to the requested object. The compatibility of the locks is shown in Table 1, where '+' indicates incompatibility.

The M_S lock is semi-preemptible. An initiator is allowed to hold an M_S lock until the lock manager asks it to release that M_S lock. A callback mechanism is used to force the holder to release the lock, when some initiator requests M_X on the same metadata object. The holding

Table 2. Compatibility of hierarchical locks

	D_S	D_X	D_IS	D_IX
D_S		+		+
D_X	+	+	+	+
D_IS		+		
D_IX	+	+		

initiator only comply the request when it has no conflicting usage of the object.

An *M_X* lock for a metadata object is requested by an initiator when it is going to modify the metadata object. The *M_X* locks will not be cached at the initiators, so initiators have to contact targets every time. An *M_X* lock is always released immediately after the involved operations have finished.

Another possible operation on an *M_S* lock is to upgrade it to a *M_X* lock. This happens when an metadata object is first read and cached locally, and later a write request for the same metadata object arrives at the same initiator.

3.2. Locking scheme for normal data

Normal data are organized into files. In order to balance between high concurrency and high resource consumption, we design a two level hierarchical locking scheme. The upper level is an entire file and the lower level contains fix-sized block groups.

There are 4 possible locks applicable to nodes of such hierarchy.

- *D_IS*: intention shared access; allowing explicitly locking on descendant nodes in *D_S* or *D_IS* mode; no implicit locking to the sub-tree.
- *D_IX*: intention exclusive access; allowing explicitly locking on descendent nodes in *D_X*, *D_S*, *D_IX*, or *D_IS* mode; no implicit locking to the sub-tree.
- *D_S*: shared access; implicit *D_S* locking to all descendants.
- *D_X*: exclusive access; implicit *D_X* locking to all descendants.

Intention mode is used to indicate that compatible locks are going to be requested at finer level and thereby prevents incompatible non-intention locks (*D_S* and *D_X*) on upper level. Table 2 gives the compatibility of lock modes, where '+' means conflict.

Locks are always requested from root to leaves. On the other hand, locks should be reversely released from leaves to root. Intention modes are not applicable to leaf nodes.

3.3. Cache consistency control

Physical blocks fetched over networks are cached in iSCSI initiators' buffer caches. Buffer caches will be checked first when a physical block is requested. In order to avoid revalidating consistency of cached data blocks every time, we employ a mechanism based on callback. A callback record will be set up on iSCSI target side when a physical block is read out. When an iSCSI initiator is going to write a physical block, it first sends a SCSI CDB with write request. The iSCSI initiator will wait for a R2T response before starting transmitting data. When an iSCSI target receives a SCSI CDB with write request, it will check callback records for the requested physical blocks. If there are outstanding callback records, callback requests will be sent to those iSCSI initiators to ask them to purge the requested physical blocks out of their buffer caches. A iSCSI target will not send R2T response until it receives confirmations for all callback requests that it sent out.

3.4. Transaction file sharing semantics

In our design, file-accessing operations are grouped into transactions. Every transaction will be assigned a unique transaction id within the session between an iSCSI initiator and an iSCSI target.

Deadlocks are going to happen since we are supporting transactions. Due to the nature of random access of file data, it is difficult to prevent deadlocks from happening. Therefore, we use deadlock detection mechanism to detect deadlock when they have happened. When detecting a deadlock, a victim transaction will be selected and rolled back. The mechanism to detect deadlocks is to find a loop among transactions and locks.

3.5. Implementation components

Figure 2 shows the architecture overview of our implementations. We insert new modules in to both iSCSI initiators and iSCSI targets. Our implementation is based on the ext2 file system. The metadata and file data stored on storage devices are intact.

In iSCSI initiators, vfs is used between the upper level system call layer and the lower level iSCSI layer. we have inserted following two modules into the kernel of iSCSI initiators.

- **iSCSI client module** is actually a modified ext2 file system module. It manages transactions and various metadata and normal data locks.
- **Initiator cache manager module** manages a dedicated buffer cache for the iSCSI client module. It supports callback mechanism to assure cache consistency.

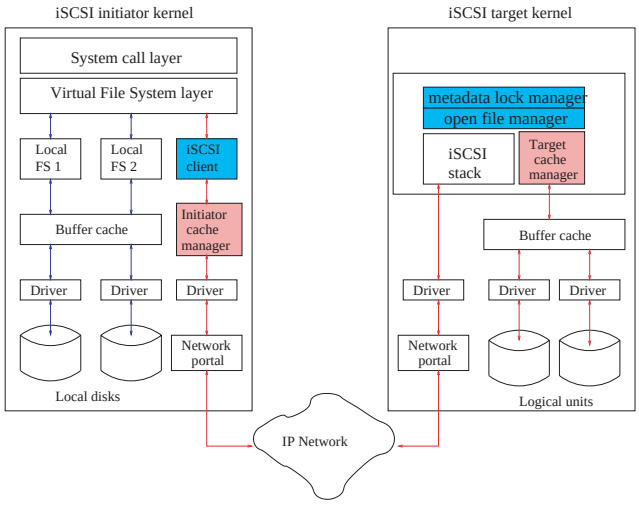


Figure 2. Overview of the architecture

An iSCSI target is responsible for maintaining active transactions, maintaining opened files, supporting callbacks for cached physical blocks, and so on. In iSCSI targets, we have inserted following three modules into iSCSI targets' kernel.

- **Metadata lock manager module** manages lock requests for metadata objects mentioned in Section 3.1. It handles deadlock detection and resolution caused by metadata locking requests.
- **File lock manager module** manages transaction requests, file open/close requests, and block group lock requests. It is also responsible for deadlock detection and resolution.
- **Target cache manager module** maintains callback records for physical blocks cached at iSCSI initiators. When there is a SCSI CDB with write command triggering callbacks, this module is also responsible for suspending the write command until all confirmations for callback requests are received.

4. Simulation Results

We use network simulator ns-2 to simulate network environments. Our schemes are implemented as modules running on host nodes in ns-2. The implementation is based on the ext2 file system as described in Section 3.5. We implement application-level iSCSI initiators and iSCSI targets, which contain components as presented in Section 3.5.

Table 3 shows the parameters we used for SCSI disk modules in all of our simulations. These parameters are following the specification of Seagate Cheetah 15K.3 family disk drives. However, no cache of disk modules is assumed in our simulations. Once a Read/Write command leaves the waiting queue on an iSCSI target for execution, the delay for Read/Write access one block of data is computed as $Delay_{Read/Write} = AverageLatency + AverageSeekTime_{Read/Write} + BlockSize/InternalTransferRate$.

[1]average of 49 to 75 Mbytes/sec

Table 3. Parameters of SCSI disk modules

Average Latency	2.0 msec
Average Read Seek Time	3.6 msec
Average Write Seek Time	3.9 msec
Internal Transfer Rate	62 Mbytes/sec[1]

In our simulations, we let iSCSI drivers on iSCSI initiators send a SCSI CDB for every physical block. iSCSI targets process requests, including locking requests and read/write requests, sequentially.

4.1. Scheme Overhead

In order to investigate the overhead of our concurrency control and cache consistency scheme, we run simulations for single sequential writing of a single file. The operations are run as a single transaction as defined in Section 3.4. The file size we used in these simulations is 100MB. For writing of each physical block, we assume that each physical block should be read from its iSCSI target first and then written back. We have run this simulation for several different network configurations. Due to space limit, we only show the result that we get under one configuration. Figure 3 shows the composition of transaction time under this configuration. We use different size for physical blocks and different size for block groups defined in Section 3.2. For certain block group size, the total transaction time will decrease as the physical block size increasing. This is because larger physical block size requires less number of transmission and hence saves propagation delay. On the other hand, when using the same physical block size and varying block group size, the time spent on normal data locks decrease as we increase block group size. Larger group size would require less number of locking requests.

4.2. Effectiveness of Caching

In our design, physical blocks are cached in iSCSI initiators' buffer cache to improve performance. Our next set of simulations is trying to understand the effectiveness of such caching as the environment of iSCSI extending from LAN to WAN. In addition, we also try to investigate what are the major factors affecting effectiveness of caching and how.

In order to reflect file access patterns of real world, we use trace data generated from modified TPC-C benchmark

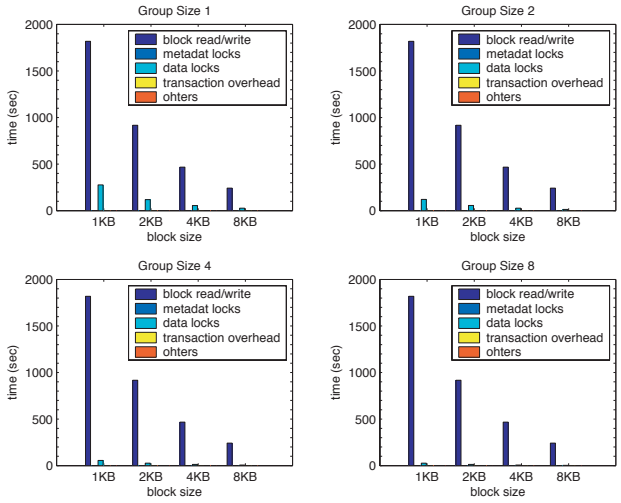


Figure 3. Composition of total transaction time for sequential access with bandwidth=100Mbps and latency=1msec

of the Transaction Processing Performance Council (TPC). The TPC-C benchmark is an OnLine Transaction Processing (OLTP) benchmark for database systems. We adapt this benchmark to generate file access traces. TPC-C benchmark involves a warehouse management database with 9 relation tables. We view each relation table as a file storing fix-sized records consecutively. TPC-C benchmark has 5 different transactions in SQL. For each transaction, we only trace the location of real reading or writing of records in the table files. We totally ignore additional database metadata such as index for keys in real database systems. For a transaction involving one or more than one table files, all table files are opened with proper mode before any reading or writing of data blocks.

Network bandwidth and latency are two potential factors that could affect effectiveness of iSCSI initiator caching. We set up a configuration of TPC-C with 4 warehouses. Each warehouse has 10 distinct districts. There are a number of customers registered to a district. Initially, we generate information to load the 9 table files. After loading initial data, the size of these 9 files range from over 380B to 120MB. We collect trace data from one client terminal, which is bond to one of the 4 warehouses. There are 200 transactions generated from this client terminal. The distribution of these transaction is 45% new order, 43% payment, 4% order status, 4% delivery, and 4% store level. In this set of simulation, we use 4K as physical block size and 1 as block group size. In Table 4, we show the comparison of with and without iSCSI initiator caching. The network bandwidth is 100Mbps and the network latency is 1msec. There are 724888 blocks accessed from cache. We also run simulations for other network configuration, which

Table 4. Comparison of w/ and w/o iSCSI initiator caching bandwidth=100Mbps latency=1msec

Time (sec)	w/ cache	w/o cache
reading blocks	29.5	5785.5
total	50.2	5806.3

show the performance will be intolerable without caching as iSCSI-based systems extend to WAN.

Our cache consistency scheme employs callback mechanism. A SCSI write command will be blocked at an iSCSI target until all response for callback requests are received. Therefore, access patterns for blocks will affect effectiveness of caching and performance of caching consistency control scheme. Still using the aforementioned TPC-C configuration, we run simulations three times with 2, 3, and 4 client terminals, respectively. Each simulation is run for 3600 seconds. In each simulation, only one client is repeatedly writing 10 physical blocks. For the other clients, they are repeatedly reading the same 10 physical blocks. Each reading and each writing is a single transaction, so no deadlock could happen. The result show that as the number of concurrent readers increases, the single writer spends more time on getting block from iSCSI target. This is caused by two reasons. First, with higher concurrency, writing command conflicts more with reading command. Secondly, with more reader, there is a higher chance that when a write command is sent, a copy of the requested block is cached on some other clients.

References

[1] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In *In Proceedings of USENIX Technical Conference*, pages 41–54, San Diego, California, June 2000.

[2] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. *iSCSI*. IP Storage Working Group, January 2003.

[3] Y. Lu and D. Du. Performance study of iscsi-based storage subsystems. *IEEE Communication Magazine*, 41(8):76–82, 2003.

[4] R. Burns, R. Rees, and D. Long. Semi-preemptible locks for a distributed file system. In *In proceedings of the 2000 International Performance Computing and Communication Conference(IPCCC)*, Phoenix, AZ, February 2000.

[5] S. Tang, Y. Lu, and D. Du. Performance study of software-based iscsi security. In *Proceedings of First International IEEE Security in Storage Workshop*, pages 70–79, 2002.

[6] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

USING DATASPACE TO SUPPORT LONG-TERM STEWARDSHIP OF REMOTE AND DISTRIBUTED DATA

Robert L. Grossman, Dave Hanley,
Xinwei Hong and Parthasarathy Krishnaswamy
Laboratory for Advanced Computing
University of Illinois at Chicago
851 South Morgan Street
Chicago, Illinois 60607
tel: +1-312-413-2176
email: grossman@uic.edu, dave@lac.uic.edu,
xwhong@lac.uic.edu, babu@lac.uic.edu

1 Introduction

In this note, we introduce DataSpace Archives. DataSpace Archives are built on top of DataSpace's DSTP servers [2] and are designed not only to provide a long term archiving of data, but also to enable the archived data to be discovered, explored, integrated and mined.

DataSpace Archives are based upon web services. Web services' UDDI and WSDL mechanisms provide a simple means for any web service client to discover relevant archived data [7]. In addition, data in DataSpace Archives can carry a variety of XML metadata, and the DSTP servers which underly the DataSpace Archives provide direct access to this metadata.

Unfortunately, web services today do not provide the scalabilty required to work with large remote data sets. For this reason, DataSpace Archives employ a scalable web service we have developed called SOAP+.

As the amount of data grows, the ability to explore and browse remote and distributed archived data will become more and more important. For this reason, a requirement of DataSpace Archives is that they support direct browsing of the data they contain, without the necessity of first retrieving the data and then opening a local application. DataSpace Archives also support a type of distributed database keys, which are described below and which enable data sets in different DataSpace Archives to be easily integrated.

Finally, DataSpace Archives use emerging internet storage platforms, such as IBP [1] and OceanStore [6], as a basis for providing long term storage, long past the demise of any individual disk or server.

2 The DataSpace Transfer Protocol (DSTP)

In previous work we have developed a protocol called the the DataSpace Transport Protocol or DSTP [2]. Data in DataSpace Archives can be accessed directly using DSTP or indirectly using web service based DSTP operations. Here is a quick summary of DSTP.

Data model. Data accessible via DSTP servers form a distributed collection of attribute-based data (in contrast to the file-based data that is usually available in data archives) that we refer to as *DataSpace*. At the simplest, data in DataSpace consists of records which may be distributed across nodes either vertically (by attributes) or horizontally (by records). Data records and data attributes are joined using universal keys, which are described next.

Universal Keys. One of the novel aspects of DataSpace is that certain attributes can be identified as universal (correlation) keys or UCKs by associating globally unique IDs or GUIDs to them. The assumption in DataSpace is that two distributed attributes having the same universal keys, as identified by their GUIDs, can be joined. This simple mechanism enables DataSpace to support vertically partitioned data, that is data records whose attributes are geographically distributed across DSTP servers.

In addition, universal keys may be attached to data sets and, in this way, data sets may be horizontally partitioned. In other words, data records may be geographically distributed across DSTP servers.

Metadata. The DataSpace infrastructure provides direct support for metadata. Each data set and each attribute can have metadata associated with it. In general, we assume that the metadata is in XML. Some DataSpace applications, with large amounts of metadata, use alternate formats for metadata for greater efficiency.

Data Access. DSTP-based access to data is via SOAP/XML or what we call SOAP+. SOAP+ is a variant of SOAP we have developed which employs a separate SOAP/XML-based control channel together with a data channel which employs a streaming protocol for moving large amounts of data or metadata efficiently. Depending upon the request, DSTP servers may return one or more attributes, one or more records, or entire data sets. DSTP servers can also return metadata about data sets or a list of universal keys associated with a data set. Each DSTP server has a special file called the catalog file containing XML metadata about the data sets on the server.

AAA Model. The default access and security model in DataSpace s a web-based instead of a grid-based access mechanism. The difference is how authentication, authorization, access (AAA) are handled. For long term stewardship, mechanisms to manage AAA are quite challenging. The default assumption in DataSpace is that data is open to anyone with a browser. Clearly, many data sets require some type of AAA infrastructure. In these cases, a AAA infrastructure can be implemented using of the standard approaches, such as Globus GSI, IETF SSL/TLS, etc.

DSTP clients and servers support the following services:

- Discovery queries. Discovery in DataSpace is via web service's WSDL and UDDI mechanisms. This provides a standards based discovery mechanism for the discovery of data sets, data attributes, metadata, etc.
- Metadata queries. DSTP servers automatically create XML metadata about the data they serve and provide a simple mechanism for user supplied metadata. DSTP Clients can request attribute based metadata, data set based metadata, or metadata summarizing all the datasets managed by the DSTP server. For example, metadata associated with an attribute typically contains the number of data records on the DSTP server associated with that attribute, a description of the attribute, the min and max values, and perhaps the providence of the attribute.
- UCK queries. Several DSTP operations are based upon universal keys or UCks. For example, a DSTP client can request all UCKs from two distributed DSTP servers, set a UCK, and then request all attributes associated with that UCK to join vertically partitioned data. Similarly, a DSTP client can request the UCKs from two distributed DSTP servers associated with data sets on the server, set a data set UCK, and retrieve all data records associated with that UCK to merge two horizontally partitioned data sets.
- Range based queries. DSTP client and servers support range based queries. Ranges may be determined using a single UCK or using several UCKs.
- Server side sampling. It is easy for DSTP Servers to overwhelm DSTP Client applications with data. The DSTP servers support server side sampling so that the appropriate amounts of data can be returned.
- Support for missing values. DSTP servers and clients support missing values as a primitive data type. This is important for exploratory data analysis and many data mining applications.

3 DataSpace Archives

We define a DataSpace Archive to consist of one or more DataSpace DSTP servers with the following the additional structure:

- *Internet-based, replicated storage.* Since we cannot guarantee over long periods of time that DSTP servers will be backed up and that the hardware will be maintained, we have decided instead to rely, in part, on distributed, replicated internet based storage, such as that provided by the OceanStore Project [6] or an Internet Backplane Protocol enabled storage system [1]. We are not suggesting that either of these two projects is ready yet to provide the type of long term storage required for archiving data, but rather that, in principle, this type of approach may play a role in helping to ensure the long term survival of data.

- *XML based metadata.* DSTP servers have a mechanism for attaching auxiliary information, such as XML metadata, to data sets. For example, PMML data may be attached to data sets in DSTP servers. The Predictive Model Markup (PMML) Language is an XML based markup language for working with statistical and data mining data sets and models [3]. It can be used to describe data attributes, data mining attributes, and some common transformations used for preparing data for analysis and mining. Maintaining data over long periods of time is made more complex by the many transformations, normalizations, and aggregations that are generally part of data prepration. The assumption in DataSpace Archives is that these are done using the transformations specified by an open XML-based standard, such as PMML.

- *Physical replicas.* In practice, one of the most durable mechanisms for preserving data is paper, as well as other durable materials such as metal or stone. Although neither practical nor desirable in many cases, it is still useful to be able to associate physical replicas with DataSpace data sets. If an ID (bar code, radio frequency ID, etc.) is attached to the physical replica, then this ID may be associated with the data set in a DataSpace Archive. More generally, GUIDS of various types may be associated with attributes and data sets in DataSpace. These can be used so that reports, laboratory books, disks, tapes, etc. can be associated with attributes, collections of attributes, and data sets in DataSpace.

4 Implementation and Experimental Studies

To understand some of the stewardship issues associated with DataSpace Archives, we have taken the data sets in the University of California at Irvine Knowledge Discovery in Databases Archive (UCI KDD Archive) [5] and created a DataSpace Archive using them. This is available on line at data.dataspaceweb.org.

Prior to the DataSpace Archive which we created, the only way to retrieve data from the UCI KDD Archive was via ftp. Moreover, as far as we are aware, the UCK KDD Archive was available on only a single server. Using a DataSpace Archive, UCI KDD data can now not only be directly viewed, but in addition, it can be explored with simple exploratory data analysis operations. In addition, XML based metadata is readily available.

We have also integrated the DSTP server with the Internet Backplane Protocol or IBP [1], which provides a baseline replication of the physical storage underlying the DSTP server. We are currently exploring additional internet based storage approaches in order to improve the long term survival of the data, such as OceanStore [6]. We have not implemented any physical replicas yet.

Finally, we note that DataSpace's support for UCKs allows UCK KDD Archive data to be integrated and correlated with other DataSpace data for the first time.

Number Records	**DSTP: SOAP/XML (sec)**	**DSTP: SOAP/XML+ (sec)**	**Speed-up**
10,000	0.65	0.21	3.10
50,000	177	0.72	245.83
150,000	673	125	5.38
375,000	3078	301	10.23
1,000,000	21121	823	25.66

Table 1: DSTP servers have two modes for data access. One mode uses SOAP/XML, which works for small data sets, and (small) metadata. The other mode uses a new protocol called SOAP+ with seperate control and data channels which scales much better for large and complex data sets.

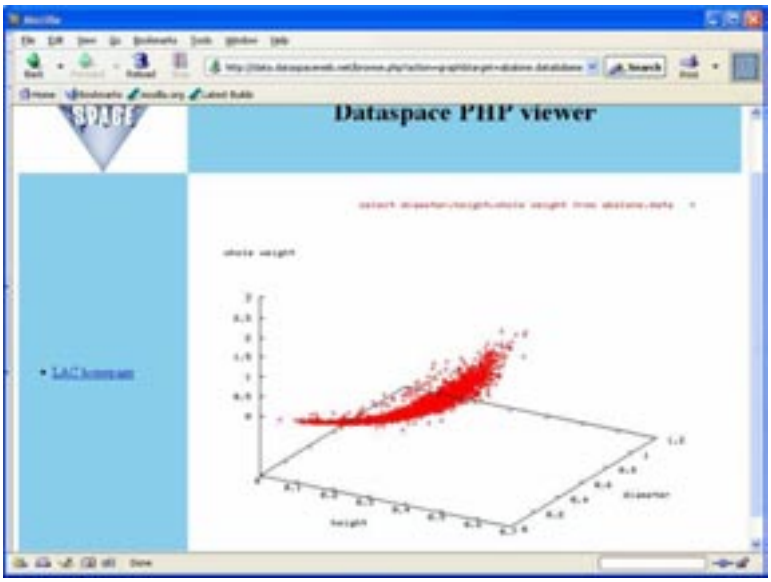


Figure 1: We have put most of the data from the UCI KDD data sets into DataSpace Archives. This figure illustrates the results of browsing one of the UCI KDD data sets and graphing the results.

5 Summary and Conclusion

In this note, we have introduced DataSpace Archives, which are archives for data sets built on top of DataSpace's DSTP servers. DataSpace Archives are designed to support the long term archiving of data in a format in which it is easy to browse and explore the archived data. In addition, DataSpace Archives enable geographically distributed data to be integrated, even if the data comes from separate data sets. In general, most current archives provide ftp access to data, but do not support the browsing, exploration, and integration of large quantities of remote and distributed data.

We have implemented DataSpace Archives on top of Version 3.0 of DataSpace and created an archive containing the University of California at Irvine Knowledge Discovery in Databases Archive.

We are currently creating a DataSpace Archive containing approximately a hundred data sets and about a Terabyte of data in order to test these ideas further.

We are also currently improving the internet based replicated storage for DataSpace Archives, as well exploring additional mechanisms to ensure that long term survival of the data is possible, such as the use of physical replicas for important data sets.

We are also working with the Data Mining Group [4] to standardize web services for data discovery, data integration, and data mining so that DataSpace Archives remain open and non-proprietary.

References

[1] M. Beck, T. Moore, and J. Plank. An end-to-end approach to globally scalable network storage. In *ACM SIGCOMM 2002 Conference*, 2002.

[2] Robert Grossman and Marco Mazzucco. Dataspace - a web infrastructure for the exploratory analysis and mining of data. *IEEE Computing in Science and Engineering*, pages 44–51, July/August, 2002.

[3] Data Mining Group. Predictive model markup language (PMML). Retrieved from http://www.dmg.org, September 30, 2003.

[4] Data Mining Group. Web services for data mining. Retrieved from http://www.dmg.org, September 30, 2003.

[5] University of California at Irvine (UCI). UCI knowledge discovery in databases archive. http://kdd.ics.uci.edu/, retrieved on September 4, 2003.

[6] Sean Rhea, Chris Wells, Patrick Eaton, Dennis Geels, Ben Zhao, Hakim Weatherspoon, and John Kubiatowicz. Maintenance-free global data storage, 2001.

[7] W3C semantic web. Retrieved from http://www.w3.org/2002/ws/, September 4, 2003.

Promote-IT: An efficient Real-Time Tertiary-Storage Scheduler

Maria Eva Lijding, Sape Mullender, Pierre Jansen
Distributed and Embedded Systems Group (DIES), University of Twente
P.O.Box 217, 7500AE Enschede, The Netherlands
{lijding, sape, jansen}@cs.utwente.nl
tel +31-53-4893770, fax +31-53-4984590

Abstract

Promote-IT is an efficient heuristic scheduler that provides QoS guarantees for accessing data from tertiary storage. It can deal with a wide variety of requests and jukebox hardware. It provides short response and confirmation times, and makes good use of the jukebox resources. It separates the *scheduling* and *dispatching* functionality and effectively uses this separation to dispatch tasks earlier than scheduled, provided that the resource constraints are respected and no task misses its deadline.

To prove the efficiency of Promote-IT we implemented alternative schedulers based on different scheduling models and scheduling paradigms. The evaluation shows that Promote-IT performs better than the other heuristic schedulers. Additionally, Promote-IT provides response-times near the optimum in cases where the optimal scheduler can be computed.

1 Introduction

Today multimedia data is generally stored in secondary storage (hard disks) and is from there delivered to the users. However, the amount of storage capacity needed for a multimedia archive is large and constantly growing with the expectations of the users. Tertiary-storage jukeboxes can provide the required storage capacity in an attractive way if the data can be accessed with real-time guarantees.

A *jukebox*[1] is a large tertiary storage device that can access data from a large number of removable storage media (RSM, for example DVDs or tapes) using a small number of drives and one or more robots to move RSM between their shelves and the drives. A central problem with this setup is that the RSM switching times are high, in the order of tens of seconds. Thus, multiplexing between files may be many orders of magnitudes slower than on a hard drive, where it takes only a few milliseconds. The second important problem is the potential for resource contention that results from the shared resources in the jukebox.

[1] We use the term *jukebox* to refer to any type of *Robotic Storage Library (RSL)*

Our *hierarchical multimedia archive (HMA)* is a service that provides flexible real-time access to data stored in tertiary storage. The HMA can serve complex requests for the real-time delivery of any combination of media files it stores. A request consists of a deadline and a set of *request units* for individual files (or part of files). Such requests can for instance result from a database query to compile a historical background for news on-the-fly, or from a personalized entertainment program consisting of music video clips. The HMA can also be used to provide real-time guarantees in the access of scientific data, e.g., earth measurements, weather forecast. In the latter cases it is especially important to be able to tell the users in advance when the data will be available in secondary storage.

Tertiary storage plays an important role in supercomputing environments and scientific computing. Essential to these environments is the capacity to deal with petabytes of data that must be easily accessible to geographically distributed scientists. The storage hierarchy that stores the data must be transparent to the users, except for the delays of accessing data in tertiary storage. The IEEE Mass Storage System Reference Model [7] describes the characteristics such systems should posses. Multiple *hierarchical storage management (HSM)* systems have been developed, both conforming to the reference model and prior to it. Some examples are the High Performance Storage System (HPSS) of the National Storage Laboratory [23], and the Storage and Archive Manager File System (SAM-FS) of Fujitsu [9]. The openness of the reference model permits to include specific real-time services as future interfaces [23]. However, no HSM so far supports real-time services. Our HMA can be incorporated in the reference model as a *Storage Server* component.

The HMA uses secondary storage as a buffer and cache for the data in its tertiary-storage jukeboxes. The *jukebox scheduler* is the key component of the HMA that guarantees the in-time promotion of data from tertiary storage to secondary storage. Apart from providing real-time guarantees, the scheduler also tries to minimize the number of rejected requests, minimize the response time for ASAP requests, minimize the confirmation time, and optimize hardware utilization.

We use a new design of jukebox schedulers, where the *scheduling* and *dispatching* functionality are clearly separated. This separation allows us to improve the performance of the system, because the optimality criteria of both functions are different. The goal of the *schedule builder* is to find feasible schedules for the requested data. Thus, the scheduler tries to build schedules as flexible as possible and is not concerned about the optimal use of the resources. The *dispatcher*, instead, is concerned about utilizing the jukebox resources in an efficient manner. We introduce the concept of *early dispatching*, by which a dispatcher can dispatch the tasks earlier than scheduled as long as the resource constraints are respected and no task misses its deadline.

The first step to build an efficient scheduler is to understand the scheduling problem thoroughly. On the one hand, we *model the hardware* and identify the parameters that define the hardware behavior. Our model is flexible and can represent any present and expected future jukebox hardware. On the other hand, we *formalize the scheduling problem* using scheduling theory so that its characteristics and complexity can be analyzed, and the problem can be classified and compared with other scheduling problems. Given the complexity

of the scheduling problem we are dealing with, there are many different ways in which it can be modeled.

The most important of these models is the *minimum switching model*, which models the problem as a *flexible flow shop* with three stages—load, read, unload. The model uses shared resources to guarantee mutual exclusion in the use of the jukebox resources. This model puts only a small restriction on the utilization of the resources, which additionally results in better use of the resources and system performance. The model requires that once an RSM is loaded in a drive, all the requested data of the RSM is read before the RSM is unloaded. Thus, the schedules that can be built with this model have a minimum number of switches.

Promote-IT is based on the minimum switching model. For every incoming request it builds a new schedule that includes all the previously scheduled request units plus the request units of the new request. It uses an efficient heuristic algorithm to find a solution to an instance of the minimum switching model on-line. Promote-IT can deal with any type of request and jukebox hardware. Additionally, it provides short response times and confirmation times, and makes good use of the jukebox resources.

We defined different scheduling strategies for Promote-IT, which vary in the way in which the jobs are added to the schedule. These strategies can be classified as *Front-to-Back* (*earliest deadline first (EDF)* and *earliest starting time first (ESTF)*) and *Back-to-Front* (*latest deadline last (LDL)* and *latest starting time last (LSTL)*). When using Front-to-Back, each job is scheduled as early as possible, while with Back-to-Front, each job is scheduled as late as possible. When using Back-to-Front, Promote-IT profits strongly from the separation of scheduling and dispatching. The scheduler creates schedules with idle times that are used by the dispatcher to dispatch tasks early. This combination proves useful in many cases, especially when the use of a shared robot is the bottleneck in the system.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents some more details about Promote-IT. Section 4 evaluates Promote-IT, comparing it's capabilities and performance with that of other schedulers. Finally, Section 5 concludes the paper.

2 Related Work

We first discuss two schedulers that can be used in a HMA. In Section 4 we compare the performance of these schedulers with that of Promote-IT. Later in this section we briefly discuss schedulers for more simple requests, schedulers with unsolved contention problems and schedulers for discrete-media.

Lau et al. [15] present an aperiodic scheduler for Video-on-Demand systems that can use two scheduling strategies: *aggressive* and *conservative*. When using the aggressive strategy each job is scheduled and dispatched as early as possible, while when using the conservative strategy each job is scheduled and dispatched as late as possible. These two strategies

are similar to the EDF and LDL strategies that we use in Promote-IT. An important difference between the strategies of Lau et al. and Promote-IT is that their strategies dispatch the tasks in the same sequence and time as assigned in the schedule. Thus, the conservative strategy performs poorly, because it leaves the resources idle, even when there are tasks that need executing. Another important difference is that their algorithm handles the jobs to include in the schedule as formed by a *read task* and a *switch task*. The switch task is scheduled as a unity, although it involves unloading the RSM loaded in the drive and loading the new RSM. Lau et al. assume that all the drives are identical and that the switching time is constant, independently of the drive and shelf involved. The former assumption is reasonable in many jukeboxes, but makes the algorithm difficult to generalize to the case with non-identical drives. The latter assumption is not reasonable in most of the large jukeboxes and forces the use of worst-case switching times when building the schedules. Using more accurate switching times provides better schedules.

Federighi et al. [8] use requests similar to those of the HMA. In their system the videos may be stored in multiple objects, with different sound tracks and subtitles corresponding to each video. The requests in their system have *soft deadlines*, e.g., the data should be available at around eight o'clock. Federighi et al. are mainly concerned about balancing the load on distributed video file servers, which are placed near the users [2]. An important difference with our approach is that, even if the requests consist of multiple objects, the playback only begins once all the objects are available at the video file servers. We refer to this type of approach as *Fully-Staged-Before-Starting (FSBS)*.

There are multiple proposals for scheduling continuous data stored in one RSM [4, 6, 5, 21, 25, 11]. The main difference among these proposals is whether the data should be fully staged before starting, streamed directly to the user, or pipelined (i.e., the data of a request can be consumed while other data of the request is being staged). We show in our work that pipelining the data is the best approach.

Various authors try to solve the problem of providing access to data contained in multiple RSM, however their schedulers suffer from unsolved contention problems [1, 16, 12, 3]. Therefore, these schedulers cannot guarantee that the real-time deadlines are always met. We analyze the faults in these schedulers in [17].

There are numerous proposals for scheduling requests for discrete data [24, 10, 22, 19, 20]. The goal of these schedulers is to minimize the average response time. In all cases, the conclusion is that as much data as possible should be read from an RSM when the RSM is loaded in the drive. These results support the minimum switching model that we use in Promote-IT.

More et al. [20] are concerned with performing queries on data that is stored in multiple tapes. Thus, a query may have multiple request units without real-time constraints. Their goal is to minimize the response time of each query. They model the scheduling problem as a two-machine flow-shop with additional constraints. In their model, the unload and load of a tape are coupled. They propose the longest transfer-time first (LtF) algorithm that for each query starts reading first the data of the sub-queries that require the longest transfer time. If there are multiple sub-queries for the same tape they use the SORT algorithm proposed by

Hillyer et al. [13] to decide the order in which the sub-queries should be read. The rationale behind the LtF algorithm is that while the data of the longest sub-query is being read, there is time to switch the tapes on the other drives and read the data corresponding to the shorter sub-queries. Through analytical analysis and simulations they show that LtF provides short response times.

In our HMA we can represent the type of requests More et al. are concerned with as a request with multiple request units with the same delta deadline (see Section 3 for details about the request of the HMA). The strategies of Promote-IT that use the latest starting time as parameter to sort the jobs build similar schedules to those of LtF for this type of requests, even if the length of the transfer is not the scheduling parameter used by Promote-IT. Given a set of RSM with the same deadline and different transfer times, the ones with longer transfer times will have earlier latest-starting-times. Thus, these strategies of Promote-IT will also schedule the RSM to begin earlier.

3 Promote-IT

A *request* r_i, which a user issues to the Hierarchical Multimedia Archive, consist of a deadline and a set of l_i request units u_{ij} for individual files (or part of files). The request can represent any kind of static temporal relation between the request units. Formally we express the user request structure in the following way:

$$r_i = (\tilde{d}_i, asap_i, \textit{maxConf}_i, \{u_{i\mathbf{1}}, u_{i\mathbf{2}}, \ldots, u_{il_i}\})$$
$$u_{ij} = (\Delta\tilde{d}_{ij}, m_{ij}, o_{ij}, s_{ij}, b_{ij})$$

The *deadline* $\tilde{d}_i$ of the request is the time by which the user must have guaranteed access to the data. The flag $asap_i$ indicates if the request should be scheduled as soon as possible. The user may specify no deadline ($\tilde{d}_i = \infty$) if the only restriction is that the request should be scheduled ASAP. The maximum confirmation time $\textit{maxConf}_i$ is the time the user is willing to wait in order to get a confirmation from the system, which indicates if the request was accepted or rejected. The system must provide a confirmation before making the data available, so $\textit{maxConf}_i \leq \tilde{d}_i$.

The relative deadline of the request unit $\Delta\tilde{d}_{ij}$ is the time at which the data of the request unit should be available, relative to the starting time of the request. The other parameters of the request unit m_{ij}, o_{ij}, s_{ij} and b_{ij} represent the RSM where the data is stored, the offset in the RSM, the size of the data, and the bandwidth with which the user wants to access the data, respectively.

The starting time of the request must not be later than its deadline, so $st_k \leq \tilde{d}_k$. If the request is ASAP, the scheduler assigns the request the earliest possible starting time st_k that will allow it to be incorporated into the system. Thus, the scheduler must find the minimum starting time st_k that makes $\mathcal{U}'$ schedulable, where $\mathcal{U}'$ is the set of request units that need scheduling. The scheduler tries different candidate starting times st_k^x and selects the earliest feasible st_k^x. assigns it the starting time corresponding to its deadline. If the

deadline of the request cannot be met, then the scheduler puts the request in the list of unscheduled requests until it can schedule it or *maxConf*$_i$ is reached and the request is rejected.

The structure of the scheduling algorithm of Promote-IT is the following:

1. Generate a candidate starting time st_k^x and update the deadline of each request unit so that $\tilde{d}_{kj} = st_k^x + \Delta\tilde{d}_{kj}$. The algorithm uses a variation of the bisection method for finding roots of mathematical functions.
2. Model $\mathcal{U}'$ as an instance of the minimum switching model. We represent the instance of the problem by the set $\mathcal{J}$ of jobs to schedule.
3. Compute the *medium schedules*. For each RSM, compute m medium schedules—one MS for each drive. Set the parameters of the duration and deadline of the read tasks $T_{\mathbf{2}j}$ to the corresponding values of the computed MS
4. Compute the resource assignment. The algorithm must incorporate each job $J_j \in \mathcal{J}$ into the schedule. If the algorithm succeeds in finding a valid resource assignment, the output of this step is a feasible schedule S^x; otherwise $S^x = \emptyset$. The pair (S^x, st_k^x) is incorporated into the list of analyzed solutions.
5. Repeat from step 1 until the bisection stop-criteria is fulfilled for the list of candidates, i.e. the time difference between the last unsuccessful and first successful candidate is smaller than a threshold.
6. Select the best solution. The best solution is the earliest candidate starting time for which step 4 could compute a feasible schedule ($\min\{st_k^x \mid S^x \neq \emptyset\}$). If there is no such st_k^x, the request r_k is placed in the list of unscheduled requests to be scheduled at a later time. Otherwise, the scheduler confirms the starting time st_k to the user and replaces the active schedule with the new feasible schedule.

In the *minimum switching model* we limit the number of jobs to one per RSM. This model requires that all the requested data from an RSM must be read before the RSM is unloaded from a drive. The processing environment of our model is a *flexible flow shop* with three stages ($FF_{\mathbf{3}}$). The first stage is to load an RSM to a drive, the second stage is to read the data from an RSM and the third is to unload the RSM. The jobs to be processed are of the form $J_j = \{T_{\mathbf{1}j}, T_{\mathbf{2}j}, T_{\mathbf{3}j}\}$, with one task for each stage.

Both the drives and robots may all have different characteristics. Therefore, the processors at each stage are modeled as *unrelated*. In the first stage there are l processors representing the l loader robots. In the second stage there are m processors representing the drives. In the third stage there are u processors representing the unloader robots. The robots in the first and third stage may have some elements in common and in the worst case all the elements will be the same: when all robots are able to load and unload. Because the robots may be limited to serve only a subset of drives and shelves, there are jobs that can be executed only in a subset of resources. In the model we indicate this by using *machine eligibility restrictions*.

The processing time of a reading task T_{2j} is determined by computing a separate schedule for all request units that are grouped into J_j. We call this schedule for an RSM a *Medium Schedule (MS)*. An MS determines in which order the data must be read once the RSM is in the drive. As the drives may be non-identical, we compute a separate MS for each drive. The optimization criterion for an MS is to maximize the time at which the RSM has to be loaded in a drive to start reading data from it, in such a way that the deadlines of the request units are met. In other words we want to determine the latest possible starting time of the read. If the RSM is already loaded in a drive, the goal is to read the requested data before the RSM must be unloaded.

In step 4 we use a branch-and-bound algorithm to prune the tree of possible assignments of jukebox resources to the jobs in $\mathcal{J}$. The branch-and-bound algorithm uses the *best-drive heuristic* to choose which drive will be tried first to schedule a job and prune from the tree the branches corresponding to drives which offer a worse solution. When pruning the tree, the algorithm may be throwing away a feasible solution that an optimal scheduler would find. But searching the whole tree of solutions is computationally unacceptable. For comparison, we have also implemented an optimal scheduler, but it can take up to several days to compute a feasible schedule for one new request, in contrast to the few milliseconds needed by Promote-IT.

The jobs are incorporated to the schedule using any of the four strategies presented in Section 1. None of the strategies is absolutely better than the others, because each strategy can find schedules that cannot be found by the others. However, ESTF performs best in most cases, and when the system load is very high it is convenient to use LDL. The difference in performance between the different strategies is small when compared with other schedulers. Therefore, in the next section we use only ESTF and LDL as representatives of Promote-IT.

4 Evaluation

To prove the efficiency of Promote-IT, we implemented alternative schedulers based on different scheduling models and scheduling paradigms. On the one hand, we designed two new schedulers: the *jukebox early quantum scheduler (JEQS)* and the *optimal scheduler*. On the other hand, we extended some heuristic schedulers proposed in the literature (see Section 2): the *extended aggressive strategy*, the *extended conservative strategy* and *Fully-Staged-Before-Starting (FSBS)*. Our extensions are able to deal with the requests used in the HMA, and with jukeboxes with different drive models and multiple robots. Furthermore, they do not assume constant switching and reading times. The extended schedulers have better properties than the original ones, while still keeping the features of the original schedulers that we consider most important to evaluate.

The *jukebox early quantum scheduler (JEQS)* is a periodic scheduler. The basic heuristic used by a periodic scheduler is to represent the requests as periodic tasks. A restriction of periodic schedulers is that they can be used only for some special use cases of HMA, as Video-on-Demand, because they are unable to deal with complex requests. Addition-

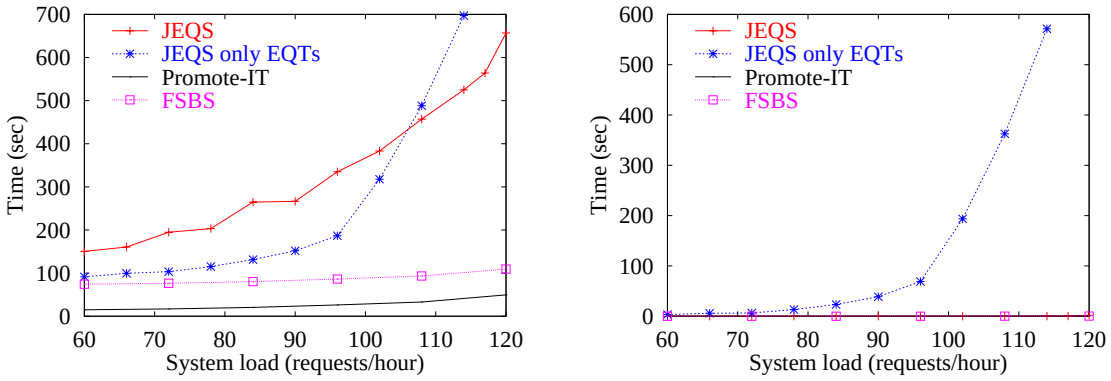
ally, periodic schedulers have serious problems in avoiding resource-contention problems. JEQS solves these problems by using the robots and drives in a cyclic way. The robot exchanges the contents of each drive at regular, fixed intervals. This results in a cyclic use of the drives, which are dedicated to reading data of an RSM while the other drives are being served by the robot. To our best knowledge, JEQS is the only correct periodic jukebox scheduler. The other periodic jukebox schedulers presented in the literature do not deal correctly with the resource-contention problem.

JEQS uses the scheduling theory on *early quantum tasks (EQT)* presented in [14]. An early quantum task is a task whose first instance is executed in the next quantum after its arrival and the rest of the instances are scheduled in a normal periodic way with the release time immediately after the first execution. Although, JEQS is generally able to start incoming requests in the next cycle of a drive, its performance is much worse than that of any of the aperiodic schedulers.

The *optimal scheduler* is a scheduler that computes the minimum response time for each incoming request. The objective of this scheduler is to be used as a baseline for evaluating the quality of the heuristic schedulers. The optimal scheduler cannot be used in a real environment due to its computing-time requirements. The computing time increases exponentially with the complexity of the requests and the system load. Therefore, we can only use it for evaluation of small test sets and relatively low system load. The comparisons that we performed show that the performance of Promote-IT is near the optimum, at least under these special testing conditions.

The simulations shown in this section were performed with JukeTools [18], using in each case representative jukebox architectures and request sets. Except when evaluating the optimal scheduler, the size of the cache is 10% of the jukebox capacity. The number of requests that can be handled by the schedulers in each of the examples shown depend on the request set and the hardware used. In each of the individual comparisons all the schedulers use the same request set and hardware simulation. As we show, some schedulers can handle load better than others, and in general the load shown in the graphics is determined by the load that can be handled by the most restrictive scheduler being compared.

Figure 1(a) compares the response time of aperiodic and periodic schedulers. Aperiodic scheduling is represented by Promote-IT and FSBS, periodic scheduling by JEQS. In this comparison the performance of Promote-IT is representative for the performance of the extended aggressive strategy and extended conservative strategy, because the difference in performance among these schedulers is negligible when compared with the difference among Promote-IT, FSBS and JEQS. For JEQS we consider two variations: scheduling normal quantum tasks (shown in the plots as 'JEQS') and scheduling only EQTs (shown in the plots as 'JEQS only EQTs'). We use FSBS in this comparison, because even though FSBS is very simple in many cases it performs better than JEQS. FSBS has a similar behavior to a First-Come-First-Serve scheduler, which virtually means that no serious scheduling is done. It first serves a request completely and only then it provides access to the data of the request.



(a) Mean Response Time

(b) Mean confirmation time.

Figure 1: Aperiodic vs. periodic scheduling.

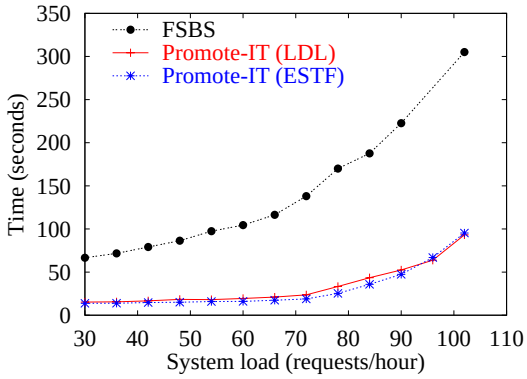


Figure 2: Pipelining vs. full staging. Mean Response Time.

The request set consists of 1000 ASAP requests that follow a Zipf distribution. Each request corresponds to one long-video file, because of the restrictions imposed by JEQS. To be able to use JEQS the request must be only for data stored in one RSM in a contiguous fashion. Additionally, JEQS needs the data to be continuous-media. When using Promote-IT the request is split in request units of 100 MB in size. The requests cannot be rejected, i.e., deadline and maximum confirmation time are infinite. The data in the jukebox is stored in double-layered DVDs and each video is stored completely in one disk. However, one disk may store multiple videos.

The jukebox has four identical DVD drives and one robot. The load time is between 21.8 and 24.8 seconds, while the unload time is between 14.3 and 17.4 seconds. The drives use CAV technology and have a transfer speed that ranges between 7.96 and 20.53 MBps and a maximum access time of 0.17 seconds.

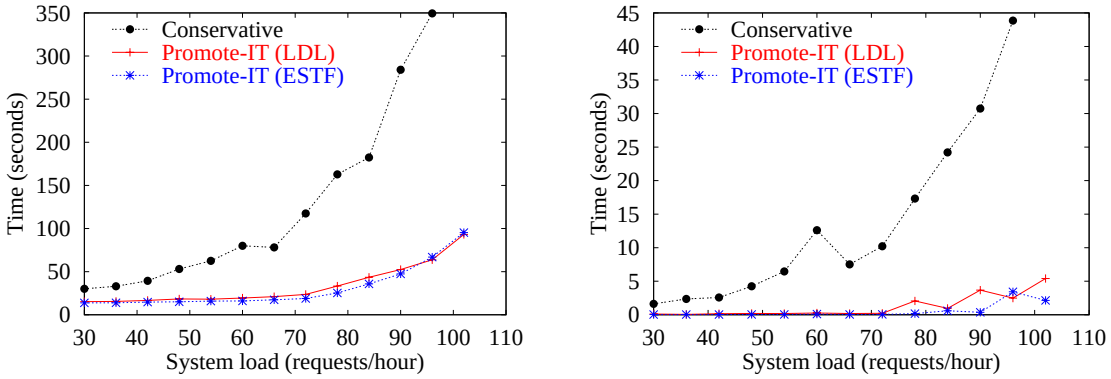
The response time of Promote-IT is much shorter than the response time of JEQS. As the system load increases, the performance of FSBS is also better than that of JEQS. JEQS uses the resources poorly, because it performs multiple switches for reading data from an RSM. In contrast, Promote-IT and FSBS use the resources efficiently by performing the minimum amount of switches required to read the data.

The confirmation time of the aperiodic schedulers is shorter that that of JEQS (see Figure 1(b)). The main difference can be seen with 'JEQS only EQTs', because this scheduler waits to accept a request until it can schedule it as an EQT. As the system load increases, the possibilities of accepting a request as an EQT diminish drastically.

Periodic schedulers have a clear advantage over aperiodic schedulers in the computing time, because they just need to evaluate a couple of formulae to decide if a request is schedulable. However, this advantage is not visible to the end user, who notices only the response time and the confirmation time. When evaluating the performance of the optimal scheduler, we will show that the computing time becomes an important parameter when it influences the confirmation time.

We conclude that periodic scheduling is a bad technique for scheduling a jukebox, because even the FSBS scheduler—which is extremely simple—in many cases performs better than JEQS. The bad performance of JEQS is not a characteristic of this particular scheduler, but is intrinsic to any periodic jukebox scheduler. A periodic scheduler either needs to use the robot in a cyclic way, or take into account the worst-case time for robot contention in the execution time of the tasks. Therefore, when using a periodic scheduler, the best-case starting time for a request that does not produce a cache-hit is the maximum time needed to switch all the drives, even if the system load is very low and all drives are idle. In the same scenario, the starting time for Promote-IT is in most cases just the time to load the RSM in the drive and read the data of the first request unit. For FSBS it is the time needed to stage all the data of the request.

The response time of Promote-IT is also shorter than that of FSBS, as can be seen in Figure 1(a). FSBS stages the whole file before giving access to the user. Therefore, the response of FSBS has as lower limit the time to buffer the whole file, while the lower limit for Promote-IT is the time to buffer the first request unit.



(a) Mean Response Time.

(b) Mean confirmation time.

Figure 3: Early vs. conservative dispatching

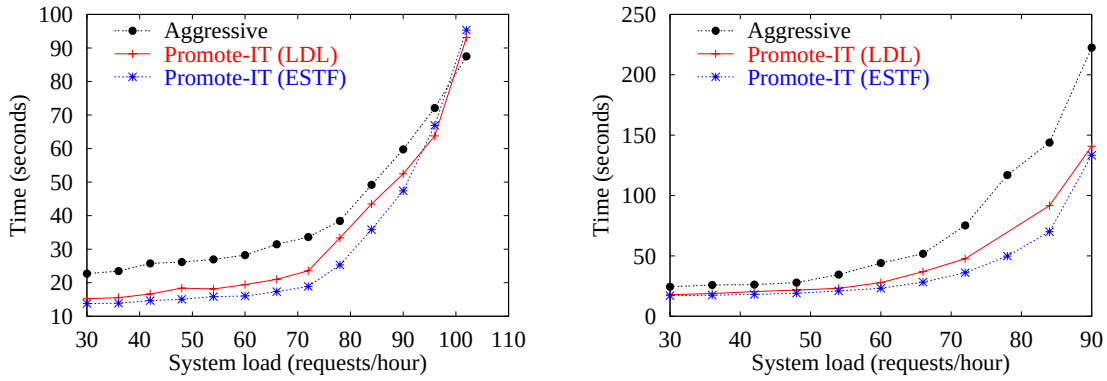
Moreover, Figure 2 shows that the difference in performance between Promote-IT and FSBS is even bigger when the data of a request is stored in multiple RSM. In this case, FSBS needs to perform multiple switches before giving access to the data, while in most cases Promote-IT only needs to perform one switch to read the data corresponding to the first request unit and the rest of the switches are performed at a later time, when the scheduler finds time for them. In this case the data in the jukebox consists of 30% long videos, 30% short videos, 30% music and 10% discrete data. The requests follow that pattern as well. The data of a request may be stored in multiple RSM.

Figure 3(a) compares the response time of Promote-IT and the extended conservative strategy (denoted as 'Conservative'). The main difference between LDL and the extended conservative strategy is the early dispatching of the tasks. As the system load increases, the difference in performance between Promote-IT and Conservative grows very fast. At the highest load level plotted, Conservative is unable to handle the load, because the waiting queue is too long. The test set is the same one described for comparing Promote-IT and FSBS with the requested data stored in multiple RSM.

The response time and confirmation time of LDL and ESTF are very similar when compared against the corresponding times of Conservative. Furthermore, when the system load is high, LDL performs slightly better than ESTF. This reinforces the idea that Back-to-Front is an interesting scheduling mechanism, when it is combined with early dispatching.

The confirmation time of Promote-IT is also lower than of Conservative (see Figure 3(b)). Conservative often fails to schedule incoming requests, because the starting time they should be assigned is too far into the future. Thus, the requests stay in the queue of unscheduled requests until the scheduler can incorporate them to the schedule.

The robot and drive utilization of Conservative is much less than that of LDL. When not using early dispatching, the resources are left idle, even if there are tasks in the schedule. Thus, when new requests arrive, their chances to be scheduled immediately are lower, even when the system load is low, because the scheduler has tasks scheduled for the future.



(a) Fast Jukebox (b) Slow Jukebox

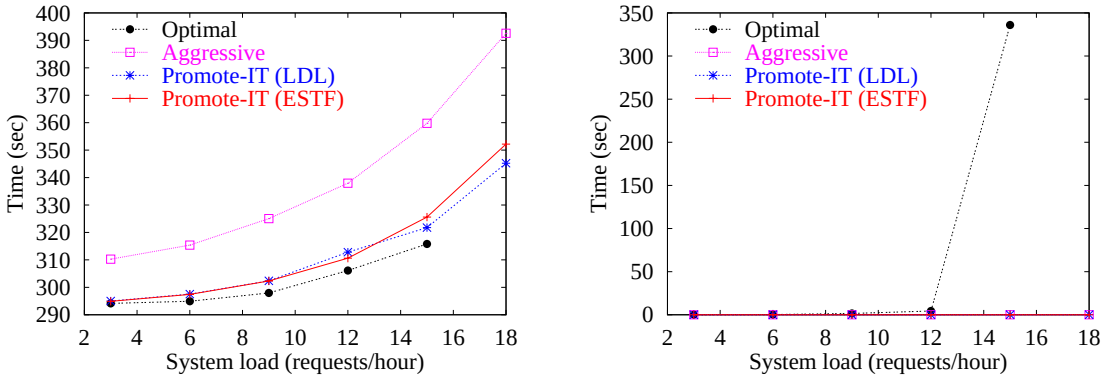
Figure 4: Uncoupled vs. Coupled load and unload. Mean Response Time

Figure 4 compares the response time of Promote-IT and the extended aggressive strategy (denoted as 'Aggressive'). The main difference between Aggressive and the ESTF strategy of Promote-IT is that Aggressive couples the load and unload into a single switch operation. This means that the RSM stay loaded in the drives until the drives are needed again. Therefore, Aggressive needs to perform first an unload before using a drive, even if the drive and the robot are idle before the request arrival.

When the system load is low and medium, Promote-IT provides shorter response times than Aggressive. However, when the system load is high and the robot is a clear bottleneck in the system, as in the case plotted in Figure 4(a), Aggressive has a better mean response time than Promote-IT. In this situation, the response time of Aggressive is similar to that of LDL, although Aggressive builds schedules Front-to-Back and LDL builds them Back-to-Front. However, Aggressive delays the last unload of a drive as much as possible, until the drive is needed again, which is the original goal of a Back-to-Front strategy. When the system load is low or medium, it is highly probable that at the time when a new request arrives there are idle resources. Therefore, delaying the unloads as much as Aggressive does affects the performance negatively. When the load is high it does not really matter, because there is no opportunity to unload the drives early anyhow. When the robot is not a strong bottleneck, as in the case plotted in Figure 4(b), Promote-IT provides shorter response times than Aggressive, even under high system loads. In this case unloading late is not beneficial: also ESTF performs better than LDL.

The request set used to compare Promote-IT and Aggressive is the same as the one used for FSBS and the extended conservative strategy. The 'Fast Jukebox' has the same configuration as previously described, in this configuration the robot is a clear bottleneck. The 'Slow Jukebox' has four DVD drives based on CLV technology with a transfer speed of 7.96 MBps and an maximum access time of 1.5 seconds.

Figure 5(a) shows that the response time provided by Promote-IT is near the optimal response time. Moreover, the difference in response time between Promote-IT and the optimal scheduler is smaller than the difference between Aggressive and Promote-IT. The plots



(a) Mean Response Time.

(b) Mean Computing Time.

Figure 5: Heuristic vs. Optimal

indicate that the difference in response time between Promote-IT and the optimal is larger as the system load increases. Therefore, we regret that we cannot run the optimal scheduler with higher loads.

The computing time of the optimal scheduler increases exponentially when the load of the system increases, while the computation time of the heuristic schedulers is nearly constant (see Figure 5(b)). The computing times of the optimal scheduler are so high that the scheduler cannot be used in an on-line system.

Additionally, we have observed that the optimal scheduler does not unload an RSM before all the data has been read in any of the runs we have performed. This is an important result in favor of the minimum switching model, on which Promote-IT is based, because even if the optimal scheduler has the possibility of performing intermediate switches, it does not do so.

The request set consists of 200 ASAP requests for long videos. The optimal scheduler does not deal with the cache administration. Therefore, each request corresponds to a different video and the cache is empty at the beginning of the runs. Thus, there are no cache-hits.

The jukebox only contains long videos, which were generated in the same way as those described in the comparison against JEQS. However, the data in the jukebox is stored in single-layered DVDs and the drives use CLV technology with a transfer speed of 6.45 MBps and a constant access time of 0.1 seconds.

When building the request sets we have to make a trade-off between keeping the number of request units per request low and having more than one request unit per RSM. The computational complexity of the optimal scheduler increases exponentially with the number of request units to schedule, so we should only split each file in a small number of request units. On the other hand, we want to give the optimal scheduler the possibility to switch an RSM without reading all the requested data from the RSM. Therefore, it is desirable to have more than one request unit per RSM. In the tests that we show here, we chose to

	FSBS	Extended Aggressive Strategy	Extended Conservative Strategy	Promote-IT	JEQS	Optimal
Flexibility: requests	++	++	++	++	−−	+
Flexibility: hardware	++	+	++	++	−	−
Response time	−−	+	−	++	−−	+++
Confirmation time	+	++	+	++	−	−−−
Computing time	++	++	++	++	+++	−−−
Deal with high load	+	++	−−	++	−−	−−−

Table 1: Summary of the performance comparison. The notation used is: excellent (+++), very good (++), good (+), bad (−), very bad (−−), and unusable (−−−).

chop the files in request units of 2.5 GB. Thus, the number of request units per request is between 1 and 4 and the number of RSM involved is 1 or 2.

Throughout this section we have shown that Promote-IT performs better than the other schedulers. However, the magnitude of the performance difference varies in each case. We put the differences in context and compare all the schedulers among each other.

We evaluate the capacity of the schedulers to deal with flexible requests and hardware. We also evaluate the schedulers regarding the response time, confirmation time, computing time and the capacity to deal with high load. Table 1 summarizes the evaluation. The classification we assigned to the schedulers in the last four categories is the result of observing their performance in multiple test setups. Although, the classification is quite subjective and difficult to quantify, we believe that it reflects correctly the average performance of the schedulers. Note that the original conservative and aggressive strategy and FSBS can only handle very limited types of jukeboxes and requests. The extensions we performed are discussed in [17].

5 Conclusions

Through Promote-IT we show that tertiary storage can be used effectively in systems with real-time requirements, for instance in a hierarchical multimedia archive. However, careful scheduling is needed in order to provide those guarantees, to use the resources efficiently, and to provide short response times to the users. A performance comparison of different schedulers, shows that Promote-IT performs better than the other heuristic schedulers, and additionally provides response-times near the optimum in cases where the optimal scheduler can be evaluated.

References

[1] J. Boulos and K. Ono. Continuous data management on tape-based tertiary storage systems. In *Proc. of the 5th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 290–301, Sept. 1998.

[2] D. W. Brubeck and L. A. Rowe. Hierarchical storage management in a distributed VoD system. *IEEE Multimedia*, 3(3):37–47, 1996.

[3] H. Cha, J. Lee, J. Oh, and R. Ha. Video server with tertiary storage. In *Proc. of the Eighteenth IEEE Symposium on Mass Storage Systems*, April 2001.

[4] S.-H. G. Chan and F. A. Tobagi. Designing hierarchical storage systems for interactive on-demand video services. In *Proc. of IEEE Multimedia Applications, Services and Technologies*, June 1999.

[5] A. L. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. PhD thesis, Dept. of Comp. Science, University of California, Berkeley, December 1994.

[6] A. L. Chervenak. Challenges for tertiary storage in multimedia servers. *Parallel Computing*, 24(1):157–176, Jan. 1998.

[7] J. L. Cole and M. E. Jones. The IEEE storage system standards working group overview and status. In *Proc. of the 14th IEEE Symposium on Mass Storage Systems*. IEEE, September 1995.

[8] C. Federighi and L. A. Rowe. Distributed hierarchical storage manager for a video-on-demand system. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 185–197, February 1994.

[9] Fujitsu. Storage management overview. White Paper, January 1998.

[10] C. Georgiadis, P. Triantafillou, and C. Faloutsos. Fundamentals of scheduling and performance of video tape libraries. *Multimedia Tools and Applications*, 18(2):137–158, 2001.

[11] S. Ghandeharizadeh and C. Shahabi. On multimedia repositories, personal computers, and hierarchical storage systems. In *Proc. of th ACM Multimedia Conference*, 1994.

[12] L. Golubchik and R. K. Rajendran. A study on the use of tertiary storage in multimedia systems. In *Proc. of Joint NASA/IEEE Mass Storage Systems Symposium*, March 1998.

[13] B. K. Hillyer and A. Silberschatz. Random I/O scheduling in online tertiary storage systems. In *Proc. of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 195–204, June 1996.

[14] P. G. Jansen, F. T. Y. Hanssen, and M. E. Lijding. Scheduling of early quantum tasks. In *15th Euromicro Conf. on Real-Time Systems*, pages 203–210. IEEE Computer Society Press, Jul 2003.

[15] S.-W. Lau and J. C. S. Lui. Scheduling and replacement policies for a hierarchical multimedia storage server. In *Proc. of Multimedia Japan 96, International Symposium on Multimedia Systems*, March 1996.

[16] S.-W. Lau, J. C. S. Lui, and P. Wong. A cost-effective near-line storage server for multimedia system. In *Proc. of the 11th International Conference on Data Engineering*, pages 449–456, March 1995.

[17] M. E. Lijding. *Real-Time Scheduling of Tertiary Storage*. PhD thesis, CTIT Ph.D.-thesis Series 03-48, Univ. of Twente, May 2003.

[18] M. E. Lijding, P. G. Jansen, and S. J. Mullender. Implementing and evaluating jukebox schedulers using JukeTools. In *20th IEEE Symp. on Mass Storage Systems*, pages 92–96, San Diego, California, Apr 2003. IEEE Computer Society Press, Los Alamitos, California.

[19] C. Moon and H. Kang. Heuristic algorithms for I/O scheduling for efficient retrieval of large objects from tertiary storage. In *Proc. of the Australiasian Database Conference*, pages 145–152. IEEE, February 2001.

[20] S. More and A. Choudhary. Scheduling queries on tape-resident data. In *Proceeding of the European Conference on Parallel Computing*, 2000.

[21] H. Pang. Tertiary storage in multimedia systems: Staging or direct access? *ACM Multimedia Systems Journal*, 5(6):386–399, November 1997.

[22] S. Prabhakar, D. Agrawal, A. E. Abbadi, and A. Singh. Scheduling tertiary I/O in database applications. In *Proc. of the 8th International Workshop on Database and Expert Systems Applications*, pages 722–727, September 1997.

[23] D. Teaff, D. Watson, and B. Coyne. The architecture of the High Performance Storage System (HPSS). In *Proc. of the Fourth NASA GSFS Conference on Mass Storage Systems and Technologies*, 1995.

[24] P. Triantafillou and I. Georgiadis. Hierarchical scheduling algorithms for near-line tape libraries. In *Proc. of the 10th International Conference and Workshop on Database and Expert Systems Applications*, pages 50–54, 1999.

[25] P. Triantafillou and T. Papadakis. On-demand data elevation in hierarchical multimedia storage servers. In *Proc. of 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 226–235, 1997.

THE DATA SERVICES ARCHIVE

Rena A. Haynes
Sandia National Laboratories, MS 0822
Albuquerque, NM 87185-5800
Tel: +1-505-844-9149
e-mail: rahayne@sandia.gov

Wilbur R. Johnson
Sandia National Laboratories, MS 1137
Albuquerque, NM 87185-5800
Tel: +1-505-845-0279
e-mail: wrjohns@sandia.gov

Abstract

As access to multi-teraflop platforms has become more available in the Department of Energy Advanced Simulation Computing (ASCI) environment, large-scale simulations are generating terabytes of data that may be located remotely to the site where the data will be archived. This paper describes the Data Service Archive (DSA), a service oriented capability for simplifying and optimizing the distributed archive activity. The DSA is a distributed application that uses Grid components to allocate, coordinate, and monitor operations required for archiving large datasets. Additional DSA components provide optimization and resource management of striped tape storage.

1. Introduction

In the ASCI environment, deployment of massively parallel computational platforms and distributed resource management infrastructure for remote access allows computations to execute on the platform that can best perform the calculation. Data generated by very large calculations can be hundreds of gigabytes to terabytes in size. Datasets are typically distributed across hundreds to thousands of files that range in size from megabytes to gigabytes. Extracting information from the data and transporting the reduced data to a local platform that can provide the interactivity required for the analysis is used to accomplish analysis and visualization of large datasets.

Large datasets typically remain in the site that generated the data until they are archived to a locally managed parallel tape storage system (HPSS [1]). Although high-speed wide area networks and parallel file transfer components allow high performance movement of data, the amount of time required to transmit large datasets allows opportunities for errors that abort the transfer. Recovery from aborted transfers requires retransmission of files. A disk file system is used to buffer the dataset files before they are transferred across the local area network to HPSS parallel tape storage. Both the wide-area and the local-area transfers cause sustained high-speed bursts of data. Until the file data has been placed onto tape storage, files remain on file system storage. Without resource control and coordination, multiple occurrences of this activity can overwhelm both the intermediate

file system and the archival system, especially as more capability platforms, like ASCI Q, Purple, and Red Storm, are deployed.

Grid middleware [2] can address some of the issues involved in archiving large distributed datasets. Core services in Grid middleware provide uniform methods for scheduling, allocating resources, and monitoring data transfer processes on distributed systems. More recently, middleware developed for Data Grids address issues involving data location, storage resources, and cache management. The SDSC Storage Resource Broker (SRB [3]) supports location transparency and data replication. Storage Resource Managers [4], [5] implement storage access and cache management policies. Current Data Grids [6], [7], [8], [9] are built to support efficient and cost effective access to large data collections for a geographically distributed community of scientists. Because of this, the focus has been on enabling distributed access to a store of metadata and data, which may also be replicated in the data grid. Our focus is on usability, robustness, and resource issues involved in transferring large datasets to a parallel tape archival storage.

We describe the Data Service Archive (DSA), a service oriented capability for simplifying and optimizing the distributed archive activity. The DSA is a distributed application that uses Grid components to allocate, coordinate, and monitor operations required for archiving large datasets. Additional DSA components provide optimization and resource management capabilities for striped tape storage. The following sections present requirements, a functional overview, and performance of the DSA application. Planned extensions to the DSA are also discussed.

2. High Level Requirements

The requirements for the DSA include the usual requirements of a simple, easily managed user interface to make archival and retrieval requests. Ease of use includes desktop access to clear interfaces for control and status as well as to a well-defined mechanism for making archive requests. The DSA should mitigate complexities of the distributed environment by providing a common interface regardless of data location and recovery from system failures where possible; however, file and resource brokering are not required. Any desktop software should have minimal prerequisites with automated installation and update procedures. Command line invocation is also required to support requests from running processes.

Resource management requirements for the DSA include managing concurrent archival requests, scheduling requests and data transfers, optimizing data transfers, and balancing the load on the file system cache and the tape storage system. Optimizing data transfers and balancing the load on the parallel tape storage system requires obtaining knowledge of network topology, tape drive resources, and striping policies of the tape system.

Software management requirements call for a clean integration with HPSS, preserving the storage system namespace and storage system policies. Accessing the storage system must be through supported mechanisms.

Additional functional requirements for the DSA include support for data integrity features, multiple storage patterns, and archive persistence management. Data integrity features should be available to verify correct wide area network transfers as well as the final image on tape storage. Storage patterns initially to be supported include requests to create an archive of tarred files as well as a directory hierarchy of files. Archive persistence introduces the notion of useful data lifetime for an archive.

3. DSA Architecture and Functional Overview

The DSA is a distributed application with components (shown in Figure 1) located on the user's desktop, a web server, the remote platform where the dataset resides, and the local data analysis cluster that is integrated with HPSS.

The DSA GUI is started from the user's desktop by accessing the Data Services DSA URL, which automatically checks for updates and optionally downloads the latest software release. The GUI allows users to define an archive request, which requires filling in a minimal amount of information required for data transfer. Features for archive formats and data integrity options may also be selected. An optional comment field is available that can be viewed in context of the archive action request. When the request is submitted for execution, DSA prompts for an identification that will be used to generate an archive identification that uniquely identifies this archive action from all others. The user may check status from the GUI or close it and check status at a later time through the information management service interface in the GUI.

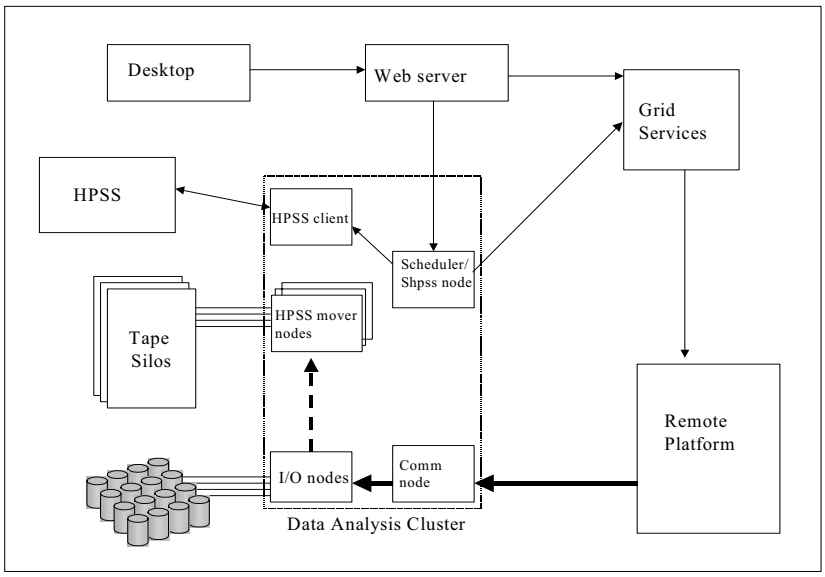


Figure 1.

Desktop software interacts with a servlet on the DSA web server to transmit the archive action request. The servlet maintains state throughout the archive process to enable recovery, restart, or cancellation. The servlet creates metadata that identifies the archive

request, then starts a two-step process to archive the data to HPSS. The servlet constructs and submits a transfer request to the scheduler on the data analysis cluster. Once scheduled, the request is sent to the grid services application server, which starts an instance of the dataset transfer program (Zephyr) on the remote platform in behalf of the user.

Zephyr, which runs with the user's credentials, determines what files are to be transferred and how the transfer is to take place. If verification is requested, a checksum is generated for each file and the checksum information is placed in the archive. If a format option is selected, the data is processed on the remote platform. After completing the information gathering and formatting steps, Zephyr transfers the data to a Parallel Virtual File System (PVFS [10]) disk storage on the data analysis cluster using a parallel file transfer protocol. This completes the first step of the archive process.

The current DSA implementation does not preallocate space on the PVFS disk cache. If a storage ceiling threshold is reached on the file system, the DSA service becomes unavailable until space is reclaimed. Dataset transfer operations detect the error and may be recovered through normal DSA recovery mechanisms.

The transfer to HPSS is scheduled immediately after completion of the transfer from the remote platform by a component called shpss. When data files are transferred as a directory hierarchy, shpss places files in groups called partitions. Files in a partition have the same approximate file characteristics (currently file size) so they will be placed in the same storage class on HPSS and, consequently, use the same tape resources. The number of files in a partition is limited to facilitate recovery if errors occur and to improve overall transfer throughput by preventing large transfers from starving small transfers in the scheduling process.

Shpss acts as tape resource manager by placing partition transfers in queues based on the partition file characteristics. The shpss queues use a node allocation mechanism to model the scheduling activity so that it tracks HPSS allocation of tape units. Use of this node allocation mechanism allows management of queues and queuing policies through standard system queue manager commands. If enough tape units are available, the transfer starts immediately. The partition transfer uses the HPSS parallel file transfer protocol interface with local file system commands to move data from PVFS on the data analysis cluster to tape over the cluster interconnect. Once files in a partition have been transferred to HPSS, they are removed from the PVFS.

DSA retrieval requests are initiated through the DSA user interface, as are archival requests. Users can request retrieval of individual files or directories of files based on the HPSS namespace to a target system and directory. Retrieval requests, like the archival requests, are processed by web service middleware that maintains state and starts a two-step process to retrieve the data from the tape storage system and place the data onto the target directory. Shpss is launched on the data analysis system to stage the data from tape storage to the PVFS disk cache. As in archival requests, shpss groups file retrievals into partitions and places each partition into a scheduling queue based on the tape resource requirements. If tape resources are available, transfers start immediately. When all data

for a retrieval request has been staged to the disk cache, the web middleware starts a Zephyr process to move the data to the target directory.

For shpss to set up partition groups for a retrieval operation, it must query HPSS to obtain information about each file in the retrieval request. Currently, only tape striping information is obtained. File media identification information is not queried. Delays can occur if files are not co-located on media or they are retrieved in a different order than stored.

The DSA gives users complete control over the portion of the HPSS namespace where their data is written. Therefore it is possible for more than one archive request to reside in a common part of the namespace (e.g. subdirectory). As HPSS performance can be affected by read requests not matching the tape file patterns created during the write process, it is possible for users to present read requests through the DSA that are less than efficient. In the ASCI environment, retrieval requests from the tape archive are made much less frequently than storage requests. Considerable thought was given to tracking tape storage patterns within the HPSS namespace. It was determined that at the present time the cost/benefit of ordering read requests to tape file images was not economical in light of the amount of work required to add the capability and the complexity introduced into the overall system. If a read request points to the 'top' of a single archive in the HPSS namespace, the file read ordering will be the same as the original write ordering

4. DSA and other Technologies

Sandia National Laboratories has had a grid in place since 2001 [11] based upon a workflow processor layered on top of the globus toolkit. Being a distributed application, the DSA exploits grid technologies by using the grid workflow processor to sequence the Zephyr and shpss components. The Globus toolkit is used for submitting partitions when scheduling transfers within shpss. Some modifications were made to the globus toolkit to enhance the operation of the DSA. Run time stdout/stderr capability was added to get immediate feedback from components running in PBS, and the ability to request DSA specific resources within the cluster where added.

As mentioned above the DSA is manipulated using servlets on a web server. This presents a well-defined API for building additional features into the system. The DSA, through the web server, can be managed either through traditional POST and GET operations through HTTPS, or by passing java objects to requisite servlets themselves. The SimTracker [12] application, being developed at the three primary weapons laboratories, is in the process of integrating DSA into its functionality through this interface.

The DSA does not use data grid technologies in the classic sense, although it is possible that gridftp [13] could be used for remote transfers. This is primarily due to a differing set of requirements between the programs. The file-tracking requirement employed in the data grid is not necessary since there is only one HPSS system where data is stored to tape and the user controls where files go in the HPSS namespace. An evaluation of

current requirements is underway that may lead to rudimentary replica management to enhance the robustness of the archive.

The ability to integrate DSA with other archiving technologies or intermediate data stores is also being examined. The infrastructure where DSA resides includes a file migration capability to assist analysts with moving data between computational resources. Each resource has its own file system(s), which are dedicated to work performed on the resource. The DSA has access to these resources and can read or write data to and from any system supporting parallel ftp.

5. Performance

Moving large data sets within the ASCI environment creates contention for infrastructure resources. The impact and severity of this contention is dependent upon the location of the data and the performance capability of the resources affected. Significant effort has been put forth to implement performance enhanced resources. High-speed networks, parallel file systems and HPSS are examples of resources where significant investment has produced high performance point solutions within the distributed architecture. However, relying on point solutions as general mechanisms for moving large amounts of data is often inefficient, impractical or requires a large investment in user effort and education.

The DSA examines three key areas that affect performance when users manually store data in the HPSS system—resource contention, storage patterns, and scheduling algorithms. HPSS integration in the computational environment looks at issues involved in resource contention in normal and failure conditions. Storage patterns affect how efficiently an archive can be placed into HPSS as well as how efficiently the tape storage is used. Scheduling algorithms impact system overhead and tape resource. HPSS schedules tape mounts within the system where it has knowledge only of the current file request and not the entire archive request. Integrating advanced scheduling with HPSS can eliminate system overhead by holding transfer sessions that cannot immediately be serviced and increase tape utilization by the algorithms used to select transfers to run.

5.1 HPSS Integration in the Computational Environment

Upon completion of a computation, data is not 'local' to the HPSS system. By local it is meant that there are systems and networks extraneous to HPSS that are involved in the movement of the computational data. These systems and networks are not dedicated to the archival process, thus are shared with other processes causing potential bottlenecks and points of failure. Contention for resource can affect the performance of data movement causing poor performance for the activity at hand as well as for other work that is contending for that resource. While poor performance can be unacceptable in the computational environment, in the worse case, resource failure can occur.

Intermittent resource failure is considered performance degradation, and must be managed by the DSA. In a manual process this is often handled by restarting the entire archival operation, as the user does not have the capability or time to manage intermediate restarts. Whether the simulation data resides on the same network as HPSS

or across the wide area, it may not be practical or possible for the user to manage the involved systems and networks. Thus the most practical method for eliminating the probability of resource failure is to reduce the number of failure points in any given process.

In order to reduce resource contention, the DSA moves data to a file system that is directly accessible by HPSS tape movers. At first glance this seems counterproductive since the data is actually moved twice. However experience has shown that increased performance in network throughput and the localizing process of data to HPSS outweighs the duplicated movement. This is especially true when the data resides at a remote location over the wide area where the network is unable to reach even modest levels of performance.

5.2 Storage Patterns

The HPSS system achieves much of its performance by striping files across multiple tape drives. This allows data to be written in parallel increasing writes speeds by a factor equivalent to the number of tape drives involved in the write process. The number of tape drives used during a write is determined by two factors:

1. The size of the file will determine a stripe width through an internal mapping defined in HPSS.
2. The ftp client may force a stripe width through a class of service request.

Both of the above features have tradeoffs in either HPSS overhead or in the ability to efficiently read data back from HPSS. The user does not want to be concerned with the details of how to effectively write data to HPSS. Furthermore, the ftp client does not lend itself to storing multiple directories of varying file sizes, some very large and some relatively small. This often leads to bulk transfers on a directory basis regardless of file size and the number of striping changes that can occur.

Using the internal mapping mechanism can cause tapes to be mounted and dismounted many times during a transfer. Long delays may be experienced due to waiting for one or more tape units to become free. In an oversubscribed HPSS system, the results can be drastic as data transfers contend for tape units. Client connections time out, small data transfers are starved by large transfers and the user must be vigilant in monitoring the entire activity to assure all their data is successfully stored.

By using the client specified mechanism, all data is stored in the same striping scheme. If the client chooses a stripe width that is too narrow, large files are written inefficiently which also affects HPSS overall throughput. If the client chooses a striping width that is too wide, small files are split across many tapes and the transfer must wait for drives to become available.

Again, in an oversubscribed HPSS system, these effects are magnified.

By using either of these methods, large data movements starve smaller ones. This has two affects on practical data storage. First, by design, the ftp client-server model can time out if asked to wait too long for an activity. Second, in the ASCI environment, security credentials can time out causing any automation of transfers to fail.

The DSA manages these issues by partitioning an entire transfer, including files in multiple directories, into data movements that share a common stripe width. Each partition is transferred one after the other at a time determined by the DSA scheduler (see below). Partitions also have finite length thus allowing the scheduling of smaller transfers between larger ones.

5.3 Scheduling Techniques

Integrating with a sophisticated system such as HPSS can be performed in two ways. The DSA can both tightly couple with internal information and state through some well-defined interface, or it can couple in a loose fashion by modeling the HPSS information and state through an external mechanism. The DSA chooses the latter in order to simplify its architecture and reduce development time. DSA transfers data using the common HPSS ftp client and models tape drive allocation using a batch queuing system scheduler.

There are two primary reasons to schedule data transfers into HPSS. First is to put on hold any ftp sessions that cannot be immediately serviced by HPSS, thus eliminating unnecessary system overhead. This situation is made worse when an active ftp connection that is transferring data alters its required HPSS resource (number of tape drives) and cannot immediately be serviced. Such situations can result in protocol timeouts and a general level of confusion on the part of the user. The second reason for scheduling data transfers is to increase HPSS utilization.

HPSS schedules tape resources in a FIFO manner when a file is opened for reading or writing. Such FIFO queuing is known to be less than optimal when there are free resources (tape drives) and a pending request that could use those resources but is not at the head of the queue.

The DSA models HPSS storage resources in a similar fashion as nodes in a cluster. The number of tape drives is managed by the queuing system and the DSA requests a specific number of tape drives when submitting a transfer. This requires the DSA to examine the size of the data to be transferred and to map that size to a particular stripe width. This also requires that data be grouped into partitions that do not violate HPSS stripe width policy for any given transfer request.

In order to optimize transfer requests, the batch queuing system scheduler is configured to use a technique called backfill. When scheduling jobs (transfers) using backfill, a standard FIFO queue is employed to determine when a job should start. However, when the job at the head of the queue cannot be started because the amount of free resources is not sufficient, the scheduler looks ‘back’ into the queue to see if any other pending job can be satisfied by the amount of free resources. If such a job exists and the amount of time required by that job is not longer than the wait time for the request at the head of the

queue, the backfill job is started. Such a technique requires that DSA calculate the amount of time required for a transfer to take place.

Determining the exact amount of time required to transfer a group of files into HPSS is not necessarily possible. While transfer rates onto tape are fairly well behaved, it is not possible to determine with high certainty, the number of tapes that will be required to service a particular transfer which effects the calculation of total transfer time. Hardware compression, the amount of existing data on a tape and the number of tape loads that must be serviced prior to any given load operation all play a roll in the level of uncertainty. The DSA uses the following equation to estimate the amount of time required to perform a given transfer:

$$T_{job} = T_{login} + \sum_{i=1}^{i=N} \left[T_{startup} + \left\lceil \left(X_i / R_{rate} \right) \right\rceil + \left(S_{width} \times T_{load} \right) \right]$$

The total time required to perform a transfer (T_{job}) equals the amount of time required to perform an ftp client login (T_{login}) plus the sum of the amount of time for each individual file to transfer. The time to transfer an individual file is a factor of some startup time for the transfer ($T_{startup}$), the file size (X_i) divided by transfer rate (R_{rate}), and the time required to perform a new load of all required tapes (S_{width}xT_{load}). It is known that this estimation is less than optimal since files can span tape volumes, which would require additional tape loads. It could be possible to obtain a fair estimate of how many tape loads a particular transfer will require, but at this point we leave such optimization to future work.

Lastly, knowing that external influences can affect a transfer and cause the time needed to move the data onto tape to exceed the requested job time, the DSA has the ability to track what files have been successfully moved. This permits ‘retrying’ files that did not move by creating a new partition containing the files that did not transfer and repeating this process.

5.4 Transfer Comparisons

A complete treatment of DSA performance is a paper unto itself. We try here to give a heads up of how well the unique features of the DSA compare to conventional manual processes for moving data. There are three questions we want to answer:

1. Does the partitioning of data files into common storage class transfers increase performance over the practice of 'mput *'?
2. Can we schedule data for transfer that will make more efficient use of HPSS resources (tape drives) and increase throughput?
3. What performance is gained when minimizing network resource utilization by staging data to a file system that is directly accessed by the HPSS movers?

To help answer these questions we ran simple experiments that transferred data into HPSS in a controlled environment. No other activity was permitted when the experiments where run. Care was taken to keep comparisons as equal as possible taking into account

that seek times would increase and deferred mounts could skew results when compared to actual mounts.

For the first question we ran ten transfers—five for an shpss run and five that performed the local file mput for pftp. All transfers used the same data set that was located on a file system mounted on the machines where the HPSS mover processes ran. Sufficient time was given between transfers to allow HPSS to settle into a common state. The data set was approximately 34 gigabytes of simulation data made up of 23 files purposefully named such that HPSS would have to request a varying number of tape drives during the pftp process.

The results were that the standard local file pftp processes averaged 15 minutes to perform the transfer while the shpss processes averaged 13 minutes. We expect this difference to grow, in favor of shpss, when the number of files and changes in class of service increases.

To examine scheduled transfers, the same considerations above were used. The HPSS system was configured with eight tape drives. Two sessions were run with each transferring nine data sets concurrently. One session started nine shpss processes in a scheduler configured to perform backfill, and one that simply ran nine different pftp sessions using mlfput. The average time taken to complete the nine pftp/mlfput transfers was 33 minutes. The average time for the nine shpss sessions to complete was 27 minutes.

To evaluate the affect of reducing network resource utilization, the first suite of tests was rerun with the standard pftp process using the normal mput command that transfers data to HPSS movers over the local area network. Results from this test showed an order of magnitude improvement in performance when the locally accessible file system cache was used.

In all, this is a brief glimpse at performance. We feel that other variables in HPSS not present during these tests could alter results. Further investigation is planned to help understand the DSA environment better and improve its performance.

6. Conclusions and Future Work

.

The focus of the DSA work to date has been on simplifying and optimizing the process of archiving large datasets. Initial testing indicates that reducing resource utilization, managing storage patterns, and scheduling transfers have accomplished performance improvements. In the near future we will be integrating the DSA service with a simulation tracking service, which maintains metadata and snapshots of results from simulation runs. We also plan to support video archiving for a video editing system. We expect these applications will require additional cache management support for retrieval operations.

Acknowledgments

The work was performed at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

References

[1] HPSS, High Performance Storage System, http://www4.clearlake.ibm.com/hpss/index.jsp .

[2] Foster, I., KesselMan, C., Tuecke, S. "The Anatomy of the Grid: Enabling Scalable Virtual Organization", The International Journal of High Performance Computing Applications, Vol. 15, 2001, pp. 200-222.

[3] Rajasekar, A., Wan, M., Moore, R. "MySRB & SRBB – Components of a Data Grid", 11th International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 24-26, 2002.

[4] Shoshani, A., Sim, A., Gu, J. "Storage Resource Managers: Middleware Components for Grid Storage", 19th IEEE Symposium on Mass Storage Systems, 2002.

[5] Shoshani, A., Bernardo, L., Nordberg, H., Rotem, D., Sim, A. "Storage Management for High Energy Physics Applications", Computing in High Energy Physics, 1998.

[6] Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S. "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", Journal of Network and Computer Applications, 23, pp. 187-200, 2001.

[7] Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., Stockinger, K. "Data Management in an International Data Grid Project", IEEE ACM International Workshop on Grid Computing (Grid 2000), December, 2000, Bangalore, India.

[8] PPDG: Particle Physics Data Grid, http://www.cacr.calteck.edu/ppdg.

[9] Tierney, B., Johnston, W., Lee, J. "A Cache-Based Data Intensive Distributed Computing Architecture for "Grid" Applications", CERN School of Computing, September 2000.

[10] Ligon, III, W.B., and Ross, R. B., "PVFS: Parallel Virtual File System," Beowulf Cluster Computing with Linux, Thomas Sterling, editor, pages 391-430, MIT Press, November, 2001.

[11] Beiriger, J., Bivens, H., Humphreys, S., Johnson, W., and Rhea, R., "Constructing the ASCI Computational Grid," Ninth IEEE International Symposium on High Performance Distributed Computing, August, 2000.

[12] SimTracker. http://www.llnl.gov/icc/sdd/img/images/SimTrack.pdf.

[13] Laszewski, G., Alunkal, B., Gawor, J., Madhuri, R., Plaszczak, P, Sun, X. "A File Transfer Component for Grids", 2003 International Conference on Parallel and Distributed Processing Techniques and Applications, June, 2003, Las Vegas, Nevada.

Interconnection Architectures for Petabyte-Scale High-Performance Storage Systems

Andy D. Hospodor
Senior Member, IEEE
andy.hospodor@ieee.org

Ethan L. Miller
Storage Systems Research Center
University of California, Santa Cruz
elm@cs.ucsc.edu

Abstract

As demand for storage bandwidth and capacity grows, designers have proposed the construction of petabyte-scale storage systems. Rather than relying upon a few very large storage arrays, these petabyte-scale systems have thousands of individual disks working together to provide aggregate storage system bandwidth exceeding 100 GB/s. However, providing this bandwidth to storage system clients becomes difficult due to limits in network technology. This paper discusses different interconnection topologies for large disk-based systems, drawing on previous experience from the parallel computing community. By choosing the right network, storage system designers can eliminate the need for expensive high-bandwidth communication links and provide a highly-redundant network resilient against single node failures. We analyze several different topology choices and explore the tradeoffs between cost and performance. Using simulations, we uncover potential pitfalls, such as the placement of connections between the storage system network and its clients, that may arise when designing such a large system.

1. Introduction

Modern high-performance computing systems require storage systems capable of storing petabytes[1] of data, and delivering that data to thousands of computing elements at aggregate speeds exceeding 100 GB/s. Just as high-performance computers have shifted from a few very powerful computation elements to networks of thousands of commodity-type computing elements, storage systems must make the transition from relatively few high-performance storage engines to thousands of networked commodity-type storage devices.

[1] A petabyte (PB) is 2^{50} bytes.

The first part of this shift is already occurring at the storage device level. Today, many storage subsystems utilize low-cost commodity storage in the form of 3.5" hard disk drives. The heads, media and electronics of these devices are often identical to storage used on desktop computers. The only remaining differentiator between desktop and server storage is the interface. At present, Fibre-Channel and SCSI remain the choice of large, high-end storage systems while the AT attachment (ATA) remains the choice of desktop storage. However, the introduction of the Serial ATA interface provides nearly equivalent performance and a greatly reduced cost to attach storage.

Most current designs for such petabyte-scale systems rely upon relatively large individual storage systems that must be connected by very high-speed networks in order to provide the required transfer bandwidths to each storage element. We have developed alternatives to this design technique using 1 Gb/s network speeds and small (4–12 port) switching elements to connect individual object-based storage devices, usually single disks. By including a small-scale switch on each drive, we develop a design that is more scalable and less expensive than using larger storage elements because we can use cheaper networks and switches. Moreover, our design is more resistant to failures—if a single switch or node fails, data can simply flow around it. Since failure of a single switch typically makes data from at least one storage element unavailable, maintaining less data per storage element makes the overall storage system more resilient.

In this paper, we present alternative interconnection network designs for a petabyte-scale storage system built from individual nodes consisting of a disk drive and 4–12 port gigabit network switch. We explore alternative network topologies, focusing on overall performance and resistance to individual switch failures. We are less concerned with disk failures—disks will fail regardless of interconnection network topology, and there is other research on redundancy schemes for massive-scale storage systems [16].

2. Background

There are many existing techniques that provide high-bandwidth file service, including RAID, storage area networks, and network-attached storage. However, these techniques cannot provide 100 GB/s on their own, and each has limitations that manifest in a petabyte-scale storage system. Rather, network topologies originally developed for massively parallel computers are better suited to construct massive storage systems.

2.1. Existing Storage Architectures

RAID (Redundant Array of Independent Disks) [2] protects a disk array against failure of an individual drive. However, the RAID system is limited to the aggregate performance of the underlying array. RAID arrays are typically limited to about 16 disks; larger arrays begin to suffer from reliability problems and issues of internal bandwidth. Systems such as Swift [9] have proposed the use of RAID on very large disk arrays by computing parity across subsections of disks, thus allowing the construction of larger arrays. However, such systems still suffer from a basic problem: connections between the disks in the array and to the outside world are limited by the speed of the interconnection network.

Storage area networks (SANs) aggregate many devices together at the block level. Storage systems are connected together via network, typically FibreChannel, through high-performance switches. This arrangement allows servers to share a pool of storage, and can enable the use of hundreds of disks in a single system. However, the primary use of SANs is to decouple servers from storage devices, not to provide high bandwidth. While SANs are appropriate for high I/O rate systems, they cannot provide high bandwidth without appropriate interconnection network topology.

Network-attached storage [6] (NAS) is similar to SAN-based storage in that both designs have pools of storage connected to servers via networks. In NAS, however, individual devices present storage at the file level rather than the block level. This means that individual devices are responsible for managing their own data layout; in SANs, data layout is managed by the servers. While most existing network-attached storage is implemented in the form of CIFS- or NFS-style file systems, object-based storage [15] is fast becoming a good choice for large-scale storage systems [16]. Current object-based file systems such as Lustre [13] use relatively large storage nodes, each implemented as a standard network file server with dozens of disks. As a result, they must use relatively high-speed interconnections to provide the necessary aggregate bandwidth. In contrast, tightly coupling switches and individual disks can provide the same high bandwidth with much less expensive, lower-speed networks and switches.

2.2. Parallel Processing

Interconnection networks for computing elements have long been the subject of parallel computing research. No clear winner has emerged; rather, there are many different interconnection topologies, each with its own advantages and disadvantages, as discussed in Section 3. Traditional multiprocessors such as the Intel Touchstone Delta [14] and the CM-5 [10] segregated I/O nodes from computing nodes, typically placing I/O nodes at the edge of the parallel computer. File systems developed for these configurations were capable of high performance [3, 12] using a relatively small number of RAID-based arrays, eliminating the need for more complex interconnection networks in the storage system.

There have been a few systems that suggested embedding storage in a multiprocessor network. In RAMA [11], every multiprocessor compute node had its own disk and switch. RAMA, however, did not consider storage systems on the scale that are necessary for today's systems, and did not consider the wide range of topologies discussed in this paper.

Fortunately, storage systems place different, and somewhat less stringent, demands on the interconnection network than parallel processors. Computing nodes typically communicate using small, low latency messages, but storage access involves large transfers and relatively high latency. As a result, parallel computers require custom network hardware, while storage interconnection networks, because of their tolerance for higher latency, can exploit commodity technologies such as gigabit Ethernet.

2.3. Issues with Large Storage Scaling

A petabyte-scale storage system must meet many demands: it must provide high bandwidth at reasonable latency, it must be both continuously available and reliable, it must not lose data, and its performance must scale as its capacity increases. Existing large-scale storage systems have some of these characteristics, but not all of them. For example, most existing storage systems are scaled by replacing the entire storage system in a "forklift upgrade." This approach is unacceptable in a system containing petabytes of data because the system is simply too large. While there are techniques for dynamically adding storage capacity to a existing system [7], the inability of such systems to scale in performance remains an issue.

One petabyte of storage capacity requires about 4096 (2^{12}) storage devices of 250 GB each; disks with this capacity are appearing in the consumer desktop market in

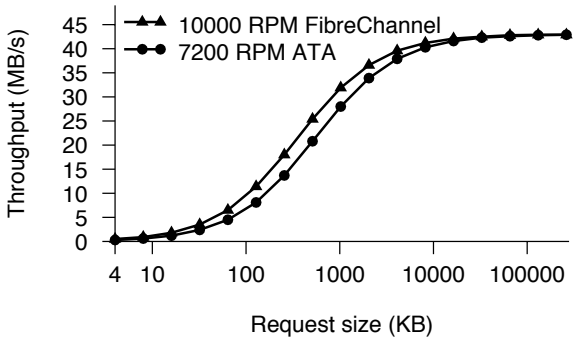


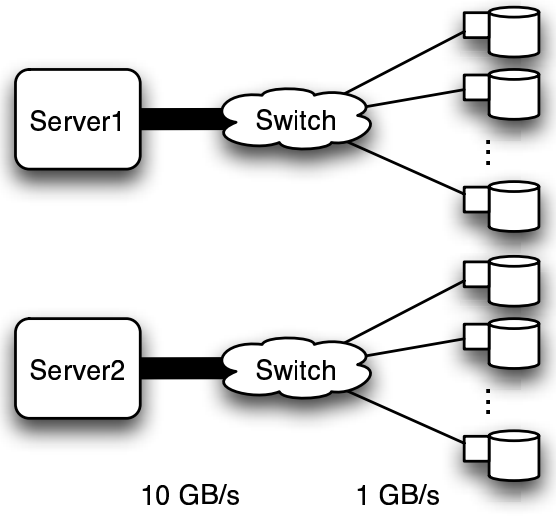
Figure 1. Disk throughput as a function of transfer size.

early 2004. A typical high-performance disk, the Seagate Cheetah 10K, has a 200 MB/s FibreChannel interface and spins at 10000 RPM with a 5.3 ms average seek time and sustained transfer rate of 44 MB/s. In a typical transaction processing environment, the Cheetah would service a 4 KB request in about 8.3 ms for a maximum of 120 I/Os per second, or 0.5 MB/s from each drive. The aggregate from all 4096 drives would be 4096×0.5 MB/s, or only 2 GB/s—far below the required bandwidth of 100 GB/s. By increasing the request size to 512 KB, disk throughput is increased to 25 MB/s per drive, for an aggregate bandwidth of 100 GB/s. Alternatively, Figure 1 shows that low-cost serial ATA drives, such as the 7200 RPM Seagate Barracuda 7200, could also meet the 100 GB/s requirement with a slightly larger request size of 1 MB.

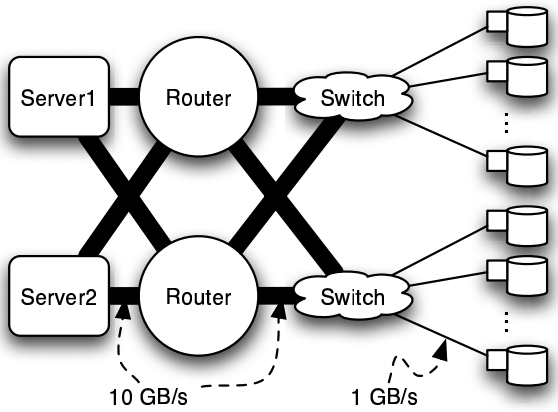
3. Interconnection Strategies

The interconnection network in a petabyte-scale storage system must be capable of handling an aggregate bandwidth of 100 GB/s as well as providing a connection to each *n*-storage node capable of transferring 25*n* MB/s of data. Thus, a 1 Gb/s network link can support nodes with at most 2–3 disks, and a 10 Gb/s network link can support up to 25 disks per node. As with any technology, however, faster is more expensive—1 Gb/s networks cost less than $20 per port, while 10 Gb/s can cost $5000 per port. In time, 10 Gb/s networks will drop in price, but it is likely that storage bandwidth demands will increase, making a tradeoff still necessary. This non-linear tradeoff in cost-performance compels us to consider complex architectures that can leverage 1 Gb/s interconnects.

The challenge facing storage system designers is an architecture that connects the storage to servers. Figure 2(a) shows a simple strategy that connects a server to 32 storage devices through a switch. Simple replication of this strategy 128 times yields a system capable of meeting the requirement. This strategy is remarkably similar to RAID



(a) Disks connected to a single server. This configuration is susceptible to loss of availability if a single switch or server fails.



(b) Disks connected to redundant servers. Switch failure is still an issue, but routers allow for more redundancy.

Figure 2. Simple interconnection strategies.

level 0, known as Just a Bunch of Disks (JBOD), and suffers from similar issues with reliability described later in the paper. Here, the port cost would be 4096 switch ports of 1 Gb/s and 128 ports of 10 Gb/s. However, the placement of data becomes crucial in order to keep all storage devices active. Since individual servers can only communicate with a small fraction of the total disk, clients must send their requests to the correct server. Unless there is a switching network comparable to that in the designs we discuss below interposed between the clients and the servers, this design is not viable. If there *is* a switching fabric between clients and servers, the designs below provide guidelines for how the network should be designed.

3.1. Fat Trees

Figure 2(b) shows a hierarchical strategy similar to a fat-tree that provides redundant connections between components. This strategy expands to have each server connect to two of eight routers that interconnect with the 128 switches that finally attach the 4096 storage devices. Each router has 32 ports attached to the servers and 128 ports attached to each of the switches and seven additional ports to the other routers. The port cost would be 4096 ports of 1 Gb/s, 2048 ports of 10 Gb/s that connect the 128 switches to the 8 routers (one port at either end), 112 ports of 10 Gb/s that interconnect the 8 routers, and 256 ports of 10 Gb/s that connect each servers to two routers. This configuration has the added drawback that the routers must be very large; it is typically not possible to build monolithic network devices with over 100 ports, so the routers would have to be constructed as multi-stage devices. While this device would allow any client to access any disk, the routers in this configuration would be very expensive. Furthermore, the 2418 ports of 10 Gb/s add nearly $10M to the overall cost, making this configuration a poor choice.

3.2. Butterfly Networks

Butterfly networks provide a structure similar to the hierarchical strategy at a more reasonable cost. Figure 3 shows a butterfly network interconnection strategy that connects disks to servers. The butterfly network can have relatively few links, but the links may need to be faster because each layer of the network must carry the entire traffic load on its links. In order to keep the individual links below 1 Gb/s, the butterfly network would need 1024 links per level for an aggregate throughput of 100 GB/s. Building a full butterfly network for 4096 disks using 1024 links and 128 switches per level would require three levels of 16-port switches and an additional level of 36-port "concentrators" to route data to and from 32 disks. Alternatively, the switching network could be built entirely from five levels of 8-port switches, using an additional level of 10-port switches to aggregate individual disks together. We use this second configuration in the remainder of the paper because, while 16 port switches are possible, we believe that the 8 port switches necessary for the second design are more reasonable.

While butterfly networks appear attractive in many ways, they do not have the fault-tolerance provided by cube-style networks such as meshes and torii. In fact, only a single path exists between any pair of server and storage devices connected by the butterfly network. In traditional parallel computers, network failures could be handled either by shutting down the affected nodes or by shutting down the entire system. Storage fabrics, on the other hand, must continue to run even in the face of network failures, making butterfly networks less attractive unless there is a method to route traffic around failed links. Cube-style networks have many routes between any two nodes in the fabric, making them more tolerant of link failures.

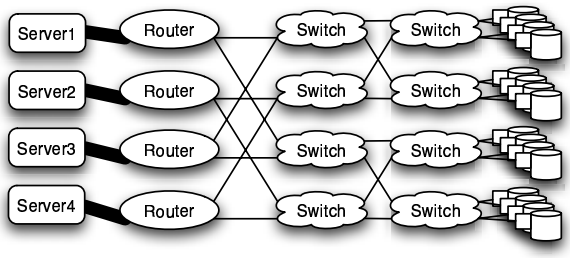


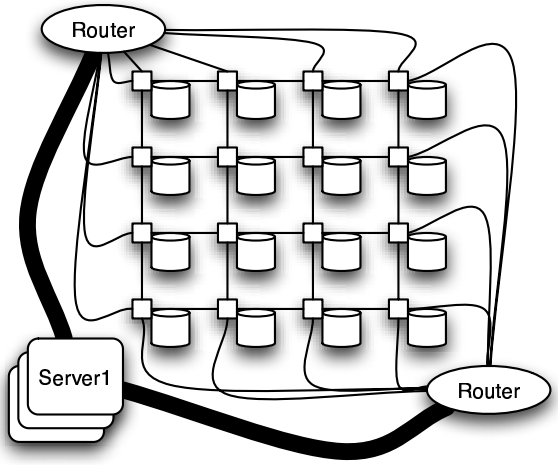
Figure 3. Disks connected in a butterfly network topology.

3.3. Meshes and Torii

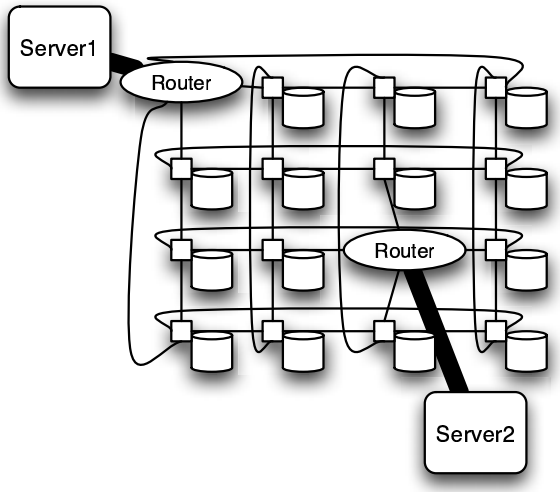
Figure 4(a) shows a mesh strategy that combines the storage device with a small switch that contains four 1 GB/s ports. The 4096 storage devices would be arranged as a 64×64 mesh with routers connecting the edge of the mesh to the servers. This configuration would require eight routers to connect to 128 servers to provide the necessary 100 GB/s bandwidth. This configuration would require 16384 1 Gb/s ports for the storage, 256 10 Gb/s ports that connect the 8 routers to the servers on two edges of the mesh, and the 256 10 Gb/s ports that connect the servers to the routers. Optionally, another 256 ports would connect all four sides of the mesh to the routers, although this much redundancy is not likely to be necessary. Router interconnects are no longer necessary because the mesh provides alternate paths in case of failure.

Torus topologies, shown in Figure 4(b), are similar to meshes, but with the addition of "wrap-around" connections between opposing edges. The inclusion of these additional connections does not greatly increase cost, but it cuts the average path length—the distance between servers and storage for a given request—by a factor of two, reducing required bandwidth and contention for network links. However, this design choice requires external connectivity into the storage fabric through routers placed at dedicated locations within the torus .

Mesh and torus topologies are likely a good fit for large scale storage systems built from "bricks," as proposed by IBM (IceCube [8]) and Hewlett Packard (Federated Array of Bricks [5]). Such topologies are naturally limited to three dimensions (six connections) per element, though they may resemble hypercubes if multiple highly connected disks are packed into a single "brick."



(a) Disks connected in a mesh topology.



(b) Disks connected in a torus topology.

Figure 4. Mesh and torus interconnection topologies.

3.4. Hypercubes

Figure 5 shows a hypercube strategy [1] of a two-dimensional hypercube of degree four that intersperses the routers and storage devices throughout the hypercube. In a 4096 node storage system, 3968 storage devices and 128 routers could be arranged in a hypercube of degree 12. Each node in this configuration has twelve 1 Gb/s ports and each router has two additional 10 Gb/s ports that connect to two servers. Bandwidth in the hypercube topology may scale better than in the mesh and torus topologies, but the cost is higher because the number of connections at each node increases as the system gets larger. Note also that hypercubes are a special case of torii; for example, a degree 12 hypercube can also be considered a 6-D symmetrical torus. Furthermore, the hypercube can be extended as a torus by adding nodes in one dimension; there is no need to add nodes in powers of two.

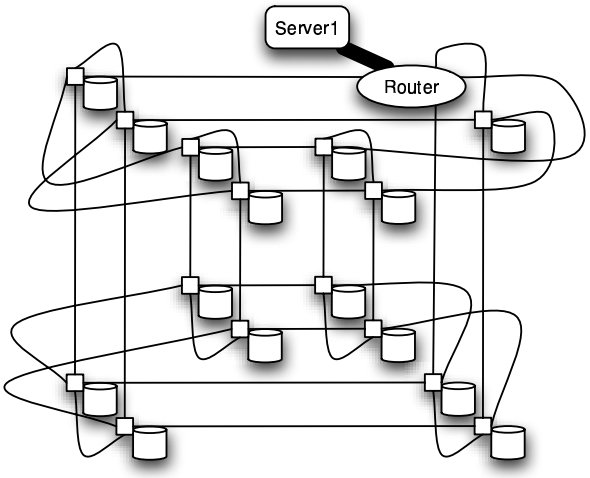


Figure 5. Disks connected in a hypercube topology.

Hypercubes and high-dimensional torii need not be built from individual disks and routers. To make packaging less expensive, a small group of disks may be assembled into a unit, and the units connected together using a lower-dimensional torus. For units with eight disks in a degree 12 hypercube, this approach requires each unit to have 48 external connections—eight connections per cube face. This is not an unreasonable requirement if the system uses gigabit Ethernet or optical fiber network connections.

4. Analytic Results

All of the topologies listed in Section 3 appear capable of providing 100 GB/s bandwidth from a cluster of 4096 disks. Further inspection shows that, because of limitations in link capacities, this is not the case. Moreover, the topologies differ in several critical ways, including overall system cost, aggregate bandwidth, latency and resistance to component failures. We analyzed the basic characteristics of seven specific topologies, listed in Table 1.

4.1. System Cost

One benefit for the designs in which switches are embedded in the "storage fabric" is that they require far fewer high speed ports at the cost of additional low speed ports. The 2004 cost of a gigabit Ethernet port is less than $20, whereas the cost of a 10 gigabit Ethernet port is on the or-

Network	Dimensions	Ports
Fat Tree	32-8-32-1024	6,512
2D mesh	64×64	16,384
2D torus	64×64	16,384
3D torus	$16 \times 16 \times 16$	24,576
4D torus	$8 \times 8 \times 8 \times 8$	32,768
5D torus	$4 \times 8 \times 4 \times 8 \times 4$	40,960
6D hypercube	$4 \times 4 \times 4 \times 4 \times 4 \times 4$	49,152
Butterfly	256 4×4 switches/layer	14,336

Table 1. Switching fabric topologies to accommodate about 4000 disks.

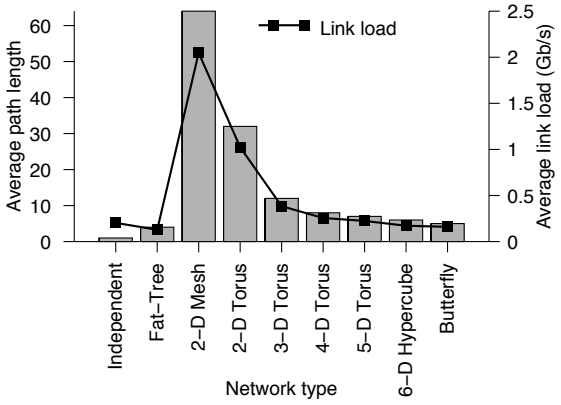


Figure 6. Average number of network hops and expected per-link bandwidth for each interconnection network topology. The "independent" topology is omitted because it relies upon the host computer for communication between storage nodes.

der of $5000. This non-linear tradeoff makes the fabric-type structures more appealing than the other structures because they simply cost less. The overall cost of a 4096 node system with different configurations is shown in Figure 7. The "independent" system is shown as a baseline; in such a system, each disk is connected to exactly one server, and servers are not connected to one another. While this is by far the least expensive option, it requires that the file system use the network of host servers to manage client access of data from the entire storage system, and limits the storage system's ability to perform internal communication for reliability and other functions. Among the other options, lower dimensionality "cubes" are the least expensive, with higher-dimension and cubes and torii being the most expensive and butterfly networks in between.

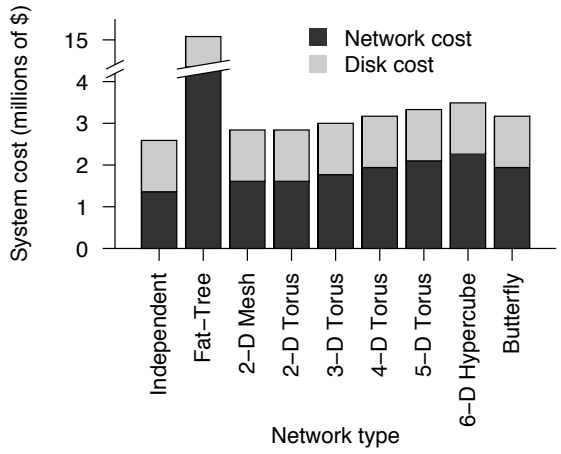


Figure 7. Total system cost for different interconnection network topologies. The "independent" topology relies upon the host computer for communication between storage nodes.

4.2. System Bandwidth

Another consideration for massive storage systems is the average number of network hops between a disk and a server, as shown in Figure 6. The number of hops is crucial because the maximum simultaneous bandwidth to all disks is $\frac{link\ speed \times number\ of\ links}{average\ hops}$. Systems with a small number of links but few average hops may perform better than systems with more links but longer average path distances between the disk and the server. For example, a $16 \times 16 \times 16$ torus might seem like a good topology because it is relatively low cost and is easy to assemble. However, this topology would have $4096 \times 6/2 = 12288$ links, and the average distance from an edge to a desired node would be $16/4 + 16/4 + 16/4 = 12$ hops. This would limit the theoretical bandwidth to $1 \times 12288/12 = 1024$ Gb/s, or 128 GB/s at most, which might be insufficient to meet the 100 GB/s demand. Figure 6 shows the expected number of hops for each network topology as well as the expected load on each network link.

Our models have assumed that links are half-duplex. If full-duplex links are used, individual links can double their theoretical maximum bandwidth. However, this doubling is only realized if the load on the link is the same in both directions. For mesh and torus topologies the load in both directions will depend on the location of the router nodes. However, for butterfly and fat-tree topologies, the ability to do full-duplex transmission is largely wasted because, for large reads, data flows in only direction through the network: from disks to routers. Large writes work similarly—data flows in one direction only, from routers to disks.

A prime consideration is the ability of individual connections into the storage fabric to supply the necessary bandwidth. For a 4096 node system supplying 100 GB/s, we have assumed that 128 external connections would be sufficient; certainly, 128 links at 10 Gb/s could indeed provide 100 GB/s of aggregate bandwidth. However, designs in which individual nodes have relatively few links cannot support such routers because the intra-fabric bandwidth available to a router is too low to support an external link of 10 Gb/s. For example, a two-dimensional fabric has four ports per node; at 1 Gb/s/port, the total bandwidth available to an external link is about 4 Gb/s, far less than the 10 Gb/s capacity of the external link and insufficient to allow 128 such nodes to provide 100 GB/s to the outside world.

Figure 6 shows that the link bandwidth required by the 4-D, 5-D and 6-D configurations could be served by 1 Gb/s interconnects, such as Gigabit Ethernet. Unfortunately, the bandwidth needs of the 2-D and 3-D configurations require the use of a faster interconnect, such as 10 Gigabit Ethernet, to meet the 100 GB/s requirement of the overall system. The cost of the necessary 10 Gb/s ports add $80M to the 2-D configuration, and a whopping $100M to the 3-D configuration.

Figures 6 and 7 make it clear that, although low-dimensionality torii are attractive because of the low number of links they require, they cannot meet the 100 GB/s requirement without resorting to more costly interconnects. On the other hand, high-dimensionality hypercubes require less bandwidth per link, yet have many links and require switches with many ports. The 4-D and 5-D torii appear to have the best combination of relatively low cost, acceptable bandwidth on individual links and reasonable path lengths. Compared to butterfly networks, the resiliency of the 4-D and 5-D torii offset the 30% to 40% added cost.

The 6-D hypercube has the highest cost and highest aggregate throughput performance. Dividing the cost by aggregate throughput, as shown in Figure 8, shows that the cost per GB/s of bandwidth is nearly identical for the butterfly and hypercube topologies. The 6-D hypercube becomes a cost effective choice for a large reliable system with quality of service constraints.

5. Simulation Results

While analytic results show theoretical performance, there are many real-world considerations that affect performance. We simulated usage of the networks whose performance was analyzed in Section 4, using a simple simulator that modeled network traffic loads along the network links. Each router between outside clients and disks made 0.5 MB requests of randomly-selected data that induced load sufficient to drive the overall system bandwidth to 100 GB/s. Requests were routed through mesh, torus, and hypercube

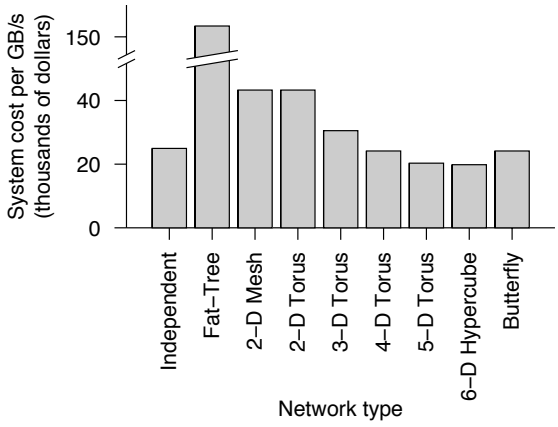


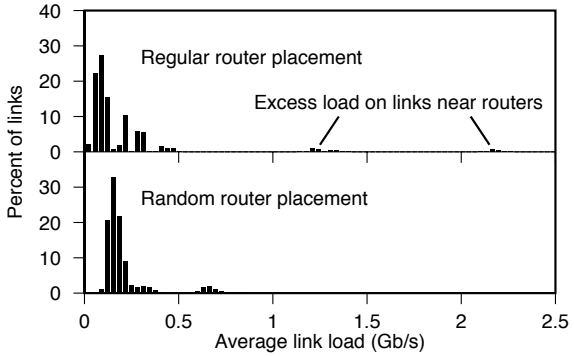
Figure 8. Cost per gigabyte per second for different interconnection network topologies.

interconnection networks using dimensional routing [4]; routing in butterfly networks is fixed because there exists only one route between requester and disk.

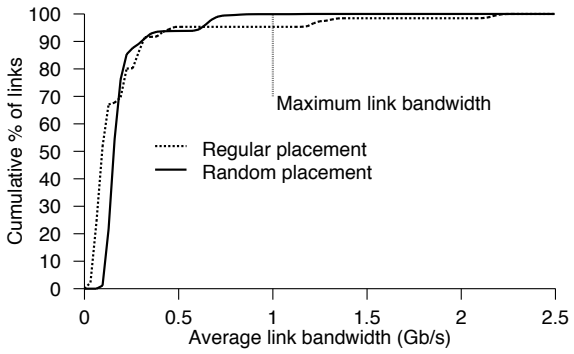
We generated a range of interconnection networks with 4096 nodes, including either 128 routers through which clients would connect to the storage fabric; the remainder of the nodes were storage nodes. The butterfly networks, on the other hand, had 4096 nodes connected through a switching fabric to 128 routers. The specific fabrics we tested are listed in Table 1.

In our first cube-style networks (meshes, torii, and hypercubes), we connected external clients through routers placed at regular locations within the network. This resulted in very poor performance due to congestion near the router nodes, as shown in Figure 9. A histogram of load distribution on individual links in a $4 \times 4 \times 4 \times 4 \times 4 \times 4$ hypercube is shown in Figure 9(a). When routers were placed in a regular arrangement, some links had bandwidths of 1.25–2.25 Gb/s. Individual disks require about 25% of a single 1 Gb/s link's bandwidth and are not affected greatly by requests that "pass through" the switch next to the disk. Routers, on the other hand, require nearly the full bandwidth of all the incoming connections. When routers are adjacent, the bandwidth is greatest nearest the routers and falls off further from the routers, resulting in overloaded links in part of the fabric and underloaded links elsewhere. Figure 9(a) shows that, in addition to overloading some links, regular placement *underloads* most of the remaining links—the histogram is shifted to the left relative to that for random node placement. The cumulative distribution of link load is shown in Figure 9(b); under regular placement, about 5% of all links experience overload.

We addressed the problem of crowding by placing routers randomly throughout the storage fabric. While this



(a) Histogram of link load.



(b) Cumulative distribution of link load.

Figure 9. Distribution of load on links in a $4 \times 4 \times 4 \times 4 \times 4$ hypercube. Randomly-placed router nodes improve the evenness of load in the fabric.

did not decrease average path length, it dramatically reduced the congestion we noticed in our original network designs, as Figure 9 shows. As a result, bandwidth was spread more evenly throughout the storage fabric, reducing the maximum load on any given link. We believe that it might be possible to further balance load by devising an optimal placement; however, this placement is beyond the scope of this paper.

6. Future Work

This paper merely scratches the surface of issues in network design for petabyte-scale storage systems. Some of the unanswered questions about this design can be answered best by building an inexpensive proof of concept system using commodity drives, gigabit networking, and small-scale switches. This setup would allow us to verify our models against a small system, providing some confidence that the large systems we are modeling will perform as expected.

As with many other computer systems, changes in the ratios between disk bandwidth and network bandwidth will also affect storage system design. For example, when 10 Gb/s network connections become inexpensive, it is likely that designs with multiple disks per switch will become feasible. Given the aggregate bandwidth limitations using 1 Gb/s links, however, placing two or three disks per switch will overload the network for most topologies.

Of course, standard issues such as protocol choice, storage system network congestion, and reliability concerns are relevant to systems in which storage is embedded in a network fabric. However, other questions such as the placement of connections into the network (edge or core), the use of a few "shortcut" links to reduce dimensionality, and other problems specific to dense interconnection networks will be relevant to designs such as those presented in this paper.

Perhaps the most important question, though, is whether this design is applicable to commercial environments, in which bandwidth is less crucial, as well as scientific computing environments. If this design is a good fit to commercial systems, it is likely that the "bricks" used to build a storage fabric will become cheaper, allowing the construction of higher performance scientific computing storage systems as well as faster, more reliable commercial storage systems.

7. Conclusions

In this paper, we have introduced the concept of building a multiprocessor-style interconnection network solely for storage systems. While this idea has been alluded to in the past, our research shows the tradeoffs between different configurations and demonstrates that "storage fabrics" based on commodity components configured as torii and hypercubes improve reliability as well as performance. More specifically, the 4-D and 5-D torii appear to be reasonable design choices for a 4096 node storage system capable of delivering 100 GB/s from 1 PB. Furthermore, these designs become faster as the system grows, removing the need to replace the entire storage system as capacity and bandwidth demands increase. It is for these reasons that we believe that storage network topologies as described in this paper will become crucial to the construction of petabyte-scale storage systems.

Acknowledgments

Ethan Miller was supported in part by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory under contract B520714. The Storage Systems Research Center is supported in part by gifts from Hewlett Packard, IBM, Intel, LSI Logic, Microsoft, Overland Storage, and Veritas.

References

[1] W. C. Athas and C. L. Seitz. Multicomputers: message-passing concurrent computers. *IEEE Computer*, 21:9–24, Aug. 1988.

[2] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2), June 1994.

[3] P. F. Corbett and D. G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, 1996.

[4] D. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[5] S. Frølund, A. Merchant, Y. Saito, S. Spence, and A. Veitch. FAB: Enterprise storage systems on a shoestring. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*, Kauai, HI, May 2003.

[6] G. A. Gibson and R. Van Meter. Network attached storage architecture. *Communications of the ACM*, 43(11):37–45, 2000.

[7] R. J. Honicky and E. L. Miller. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In *Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, NM, Apr. 2004. IEEE.

[8] IBM Company. IceCube – a system architecture for storage and Internet servers. http://www.almaden.ibm.com/StorageSystems/autonomic_storage/CIB_Hardware/.

[9] D. D. E. Long, B. R. Montague, and L.-F. Cabrera. Swift/RAID: A distributed RAID system. *Computing Systems*, 7(3):333–359, 1994.

[10] S. J. LoVerso, M. Isman, A. Nanopoulos, W. Nesheim, E. D. Milne, and R. Wheeler. *sfs*: A parallel file system for the CM-5. In *Proceedings of the Summer 1993 USENIX Technical Conference*, pages 291–305, 1993.

[11] E. L. Miller and R. H. Katz. RAMA: An easy-to-use, high-performance parallel file system. *Parallel Computing*, 23(4):419–446, 1997.

[12] N. Nieuwejaar and D. Kotz. The Galley parallel file system. In *Proceedings of 10th ACM International Conference on Supercomputing*, pages 374–381, Philadelphia, PA, 1996. ACM Press.

[13] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, July 2003.

[14] R. Stevens. Computational science experiences on the Intel Touchstone DELTA supercomputer. In *Proceedings of Compcon '92*, pages 295–299. IEEE, Feb. 1992.

[15] R. O. Weber. Information technology—SCSI object-based storage device commands (OSD). Technical Council Proposal Document T10/1355-D, Technical Committee T10, Aug. 2002.

[16] Q. Xin, E. L. Miller, D. D. E. Long, S. A. Brandt, T. Schwarz, and W. Litwin. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 146–156, Apr. 2003.

OBFS: A File System for Object-based Storage Devices

Feng Wang, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long
Storage System Research Center
University of California, Santa Cruz
Santa Cruz, CA 95064
{cyclonew, sbrandt, elm, darrell}@cs.ucsc.edu
tel +1-831-459-4458
fax +1-831-459-4829

Abstract

The object-based storage model, in which files are made up of one or more data objects stored on self-contained Object-Based Storage Devices (OSDs), is emerging as an architecture for distributed storage systems. The workload presented to the OSDs will be quite different from that of general-purpose file systems, yet many distributed file systems employ general-purpose file systems as their underlying file system. We present OBFS, a small and highly efficient file system designed for use in OSDs. Our experiments show that our user-level implementation of OBFS outperforms Linux Ext2 and Ext3 by a factor of two or three, and while OBFS is 1/25 the size of XFS, it provides only slightly lower read performance and 10%–40% higher write performance.

1. Introduction

Object-based storage systems represent files as sets of objects stored on self-contained Object-Based Storage Devices (OSDs). By distributing the objects across many devices, these systems have the potential to provide high capacity, throughput, reliability, availability and scalability. We are developing an object-based storage system with a target capacity of 2 petabytes and throughput of 100 gigabytes per second. In this system as, we expect, in many others, files will be striped across OSDs. The stripe unit size of the system will determine the maximum object size and will be the most common object size in the system. Because files will generally consist of many objects and objects will be distributed across many OSDs, there will be little locality of reference within each OSD. The workload presented to the OSDs in this system will be quite different from that of general-purpose file systems. In object-based systems that do not employ this architecture we can still expect that files will be distributed across multiple objects, objects will be distributed across multiple OSDs, and there will be little locality of reference. Even so, many distributed file systems employ general-purpose file systems as their underlying file system.

We present OBFS, a very small, highly efficient object-based file system developed for use in OSDs in large-scale distributed storage systems. The basic idea of OBFS is to optimize the disk layout

based on our knowledge of the workload. OBFS uses two block sizes: small blocks, equivalent to the blocks in general-purpose file systems, and large blocks, equal to the maximum object size, to greatly improve the object throughput while still maintaining good disk utilization. OBFS uses *regions* to collocate blocks of the same size, resulting in relatively little fragmentation as the file system ages. Compared with Linux Ext2 and Ext3 [3, 28], OBFS has better data layout and more efficiently manages the flat name space exported by OSDs. Although developed for a workload consisting mostly of large objects, OBFS does well on a mixed workload and on a workload consisting of all small objects. Thus, in addition to being highly suitable for use in high-performance computing environments where large files (and hence objects) dominate, we believe that it may also prove effective in general-purpose computing environments where small files dominate.

Our results show that our user-level implementation of OBFS outperforms Linux kernel implementations of Ext2 and Ext3 by a factor of 2 to 3, regardless of the object size. Our user-level implementation of OBFS is a little slower than a kernel implementation of XFS [19, 27] when doing object reads, but has 10% to 40% better performance on object writes. We expect the performance to improve further once we have fully implemented OBFS in the kernel to avoid extra buffer copies.

OBFS is significantly smaller than Linux XFS, using only about 2,000 lines of code compared with over 50,000 lines of code in XFS. This factor of 25 size difference and the corresponding simplicity of OBFS make OBFS easy to verify, maintain, modify, and port to other platforms. OBFS also provides strong reliability guarantees in addition to high throughput and small code size; the disk layout of OBFS allows it to update metadata with very low overhead, so OBFS updates metadata synchronously.

2. Background

A new generation of high-performance distributed file systems are being developed, motivated by the need for ever greater capacity and bandwidth. These file systems are built to support high-performance computing environments which have strong scalability and reliability requirements. To satisfy these requirements, the functionality of traditional file systems has been divided into two separate logical components: a *file manager* and a *storage manager*. The file manager is in charge of hierarchy management, naming and access control, while the storage manager handles the actual storage and retrieval of data. In large-scale distributed storage systems, the storage manager runs on many independent storage servers.

Distributed object-based storage systems, first used in Swift [6] and currently used in systems such as Lustre [4] and Slice [1], are built on this model. However, in object-based systems the storage manager is an object-based storage device (OSD or OSD) [30], which provides an object-level interface to the file data. OSDs abstract away file storage details such as allocation and scheduling, semi-independently managing all of the data storage issues and leaving all of the file metadata management to the file manager.

In a typical instance of this architecture, a metadata server cluster services all metadata requests, managing namespace, authentication, and protection, and providing clients with the file to object mapping. Clients contact the OSDs directly to retrieve the objects corresponding to the files they wish to access. One motivation behind this new architecture is to provide highly-scalable aggregate bandwidth by directly transferring data between storage devices and clients. It eliminates the file server as a bottleneck by offloading storage management to the OSDs [8] and enables load balancing and high performance by striping data from a single file across multiple OSDs. It also enables

high levels of security by using cryptographically secured capabilities and local data security mechanisms.

Much research has gone into hierarchy management, scalability, and availability of distributed file systems in projects such as AFS [18], Coda [11], GPFS [22], GFS[26] and Lustre [4], but relatively little research has been aimed toward improving the performance of the storage manager. Because modern distributed file systems may employ thousands of storage devices, even a small inefficiency in the storage manager can result in a significant loss of performance in the overall storage system. In practice, general-purpose file systems are often used as the storage manager. For example, Lustre uses the Linux Ext3 file system as its storage manager [4]. Since the workload offered to OSDs may be quite different from that of general-purpose file systems, we can build a better storage manager by matching its characteristics to the workload.

File systems such as Ext2 and Ext3 are optimized for general-purpose Unix environments in which small files dominate and the file sizes vary significantly. They have several disadvantages that limit their effectiveness in large object-based storage systems. Ext2 caches metadata updates in memory for better performance. Although it flushes the metadata back to disk periodically, it cannot provide the high reliability we require. Both Ext3 and XFS employ write-ahead logs to update the metadata changes, but the lazy log write policy used by both of them can still lose important metadata (and therefore data) in some situations.

These general-purpose file systems trade off the reliability for better performance. If we force them to synchronously update object data and metadata for better reliability, their performance degrades significantly. Our experimental results shows that in synchronous mode, their write throughput is only several MB/second. Many general-purpose file systems such as Ext2 and Ext3 use flat directories in a tree-like hierarchy, which results in relatively poor searching performance for directories of more than a thousand objects. XFS uses B+-Trees to address this problem. OBFS uses hash tables to obtain very high performance directory operations on the flat object namespace.

In our object-based storage system as, we expect, in many others, RAID-style striping with parity and/or replication is used to achieve high performance, reliability, availability, and scalability. Unlike RAID, the devices are semi-autonomous, internally managing all allocation and scheduling details for the storage they contain. The devices themselves may use RAID internally to achieve high performance. In this architecture, each stripe unit is stored in a single object. Thus, the maximum size of the objects is the stripe unit size of the distributed file system, and most of the objects will be this size. At the OSD level, objects typically have no logical relationship, presenting a flat name space. As a result, general-purpose file systems, which are usually optimized for workloads exhibiting relatively small variable-sized files, relatively small hierarchical directories, and some degree of locality, do not perform particularly well under this workload.

3. Assumptions and Design Principles

Our OBFS is designed to be the storage manager on each OSD as part of a large-scale distributed object-based storage system [13], which is currently being developed at the University of California, Santa Cruz, Storage System Research Center (SSRC). Our object-based storage system has three major components, the Metadata Server Cluster (MDSC), the Client Interface (CI), and the Storage Managers (SMs). File system functionality is partitioned among these components. The MDSC is in charge of file and directory management, authentication and protection, distributing workload among OSDs, and providing redundancy and failure recovery. The CI, running on the client

machines, provides the file system API to the application software running on the client nodes, communicates with the MDSC and SMs, and manages a local file system cache. The SMs, running on the OSDs, provide object storage and manage local request scheduling and allocation.

The operation of the storage system is as follows: Application software running on client machines make file system requests to the CIs on those machines. The CIs preprocess the requests and query the MDSC to open the files and get information used to determine which objects comprise the files. The CIs then contact the appropriate SMs to access the objects that contain the requested data, and provide that data to the applications.

In our system, objects are limited by the stripe unit size of the system. Thus, in contrast to a file, whose size may vary from bytes to terabytes, the size variance of an object is much smaller. Moreover, the delayed writes in the file cache at the client side will absorb most small writes and result in relatively large object reads and writes. We provide a more detailed analysis of the object workload characteristics in Section 4.

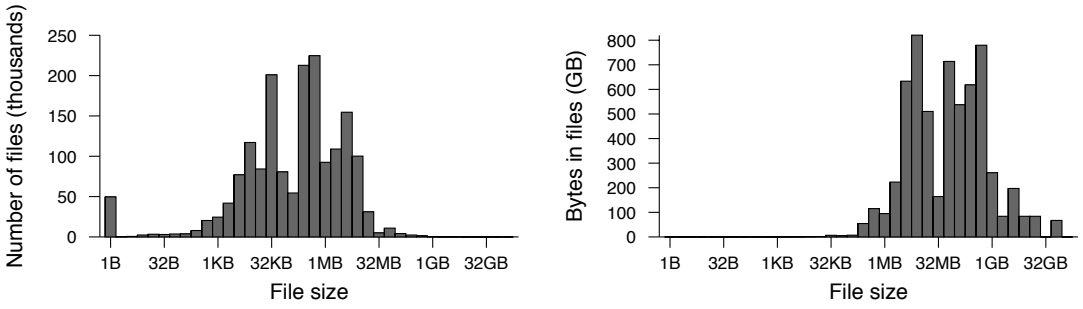
To enable parallel I/O, files are striped into fixed size objects and spread across different OSDs. The specific OSDs are selected based on the overall workload distribution intended to avoid ”hot spots” and increase potential parallelism [13]. From the viewpoint of a single OSD, incoming object accesses will be relatively random. Thus inter-object locality will be insignificant.

Most file systems cache writes for fast response, to coalesce many small writes into fewer larger ones, and to allow the file system to exploit locality of reference within the request stream. In object-based storage systems, most asynchronous writes will therefore be cached by the client. As a result, almost all of the writes to the OSDs will be synchronous. Thus, the SMs should probably not cache incoming writes in the OSDs. Furthermore, because logically contiguous data is distributed across many objects in many different OSDs, there is no locality of reference to be leveraged by caching writes of different objects.

Another caching-related concern arises due to the black-box nature of the OSDs. Because the OSDs provide a very high-level interface to the data, caching can cause the storage system as a whole to believe that the data has been saved, while data has actually been lost due to power failure or other hardware failures. While this may be addressable, we have not addressed it in this version of OBFS.

On each OSD there is a complete lack of information about relationships between objects. Thus a flat name space is used to manage the objects on each OSD. Because hundreds of thousands of objects might coexist on a single OSD, efficient searching in this flat name space is a primary requirement for the SMs.

As mentioned above, most of the incoming write requests will be synchronous. A client expects the data to be on the permanent storage when it commits its writes. This requires the OSDs to flush the objects to permanent storage before committing them. This also means the metadata of those objects should also be kept safely. In effect, OSDs in object-based storage systems are like disks in traditional storage systems, and file systems expect disks to actually store committed write requests rather than caching them.



(a) Number of files of different sizes

(b) Total bytes in files of different sizes

Figure 1. Data distribution in a large high-performance distributed storage system (data courtesy of LLNL)

4. Workload Characteristics

Very large-scale distributed file systems may exhibit very different performance characteristics than general-purpose file systems. The total volume of a large-scale distributed file system may range from several terabytes to several petabytes, orders of magnitude larger than typical general-purpose file systems. The average file size in such a distributed file system may also be much larger than that of current general-purpose file systems. Although our intent is to develop a flexible and general file system applicable in many different situations, one of our performance goals is to handle workloads encountered in high-performance computing environments with tens or hundreds of thousands of processors simultaneously accessing many files in many directories, many files in a single directory, or even a single file. These environments place extremely high demands on the storage system.

Figure 1 shows the data distribution across files in a high-performance distributed file system from Lawrence Livermore National Laboratory (LLNL). Figure 1(a) shows the file size distribution for the more than 1.5 million files in this system. Most of the files are larger than 4 KB and the majority of all files are distributed between 32 KB and 8 MB. Those files that are smaller than 4 KB (a typical block size for a general-purpose file system) only account for a very small portion of the total files. However, almost all of the disk space is occupied by files larger than 4 MB and the majority of all bytes are in files between 4 MB and 1 GB, as shown in Figure 1(b). The total number of bytes in files smaller than 256 KB is insignificant. Though the files larger than 1 GB are only a small percentage of the files, the total number of bytes in such files account for more than 15% of the bytes in the system.

The file access pattern of such systems is also different from that of a typical general-purpose file system. In the LLNL workload, most data transferred between the processors and the file system are in several megabyte chunks. Most files are accessed simultaneously by hundreds of processors. and instead of flushing dirty data directly back to the storage device, each processor caches the data in its local memory and only writes the data once the buffer is full.

Object-based storage may be used for smaller file systems as well. Systems like those traced by Roselli, *et al.* [20] have many small files; in the systems they studied, 60–70% of the bytes transferred were from files smaller than 512 KB. Clearly, an OSD file system must also be able to efficiently handle workloads composed primarily of small objects.

For the OSDs to achieve the high throughput required of the system and to fully take advantage of the object-based storage model, our system stripes file data across the OSDs. This is a very compelling choice, analogous to that of earlier systems such as Swift [6] and Zebra [9], and we believe that this will be an architecture of choice in large-scale object-based storage systems. In such systems, each object stored on an OSD will be a stripe unit (or partial stripe unit) of data from a file.

The system stripe unit size depends on the design requirements of the individual system. Stripe units that are too small will decrease the throughput of each OSD while stripe units that are too large will decrease the potential parallelism of each file. Assuming a stripe unit size of 512 KB, large files will be divided into several 512 KB objects and files smaller than 512 KB will be stored in a single object. Consequently, no object in the system will ever exceed the system stripe unit size. In the LLNL workload we estimate that about 85% of all objects will be 512 KB and 15% of all objects will be smaller than 512 KB. We will refer to objects that are the same size as the system stripe unit size as *large objects* and the rest as *small objects.* Workstation workloads [20] will likely have more small objects and fewer large objects.

Because the object-based storage system is expected to spread the objects evenly across all of the OSD devices, the object size distribution in the workload of a single OSD device will be the same as that of the larger storage system. Thus, a single OSD device under the LLNL workload should expect that 85% of incoming objects are large objects and the rest are small objects. Since files are distributed across many OSDs and directory hierarchies are managed above the OSD level, there is no inter-object locality that can be exploited in the OSDs. The workload of OSDs in this type of system will be dominated by large fixed-size objects exhibiting no inter-object locality. Under workstation workloads, in contrast, the object size distribution will be closer to 25% large objects and 75% small objects. An OSD file system should be able to handle either type of workload.

5. Design and Implementation

As described in Section 4, the expected workload of our OSDs is composed of many objects whose sizes range from a few bytes up to the file system stripe unit size. Therefore, OBFS needs to optimize large object performance to provide substantially higher overall throughput, but without overcommitting resources to small objects. Simply increasing the file system block size can provide the throughput needed for large objects, but at the cost of wasted storage space due to internal fragmentation for small objects. For the LLNL workload, more than 10% of the available storage space would be wasted if 512 KB blocks are used, while less than 1% of the space would be lost if 4 KB blocks are used. In a 2 PB storage system, this 9% difference represents about 18 TB. The situation is even worse for a workstation file system, where 512 KB blocks would waste more than 50% of the space in such a system.

To use large blocks without wasting space, small objects must be stored in a more efficient way. OBFS therefore employs multiple block sizes and uses *regions,* analogous to cylinder groups in FFS [15], to keep blocks of the same size together. Thus, the read/write performance of large objects can be greatly improved by using very large blocks, while small objects can be efficiently stored using small blocks.

Another important feature of OBFS is the use of a flat name space. As the low-level storage manager in an object-based distributed file system, OBFS has no information about the logical relationship among objects. No directory information is available and no useful locality information is likely to

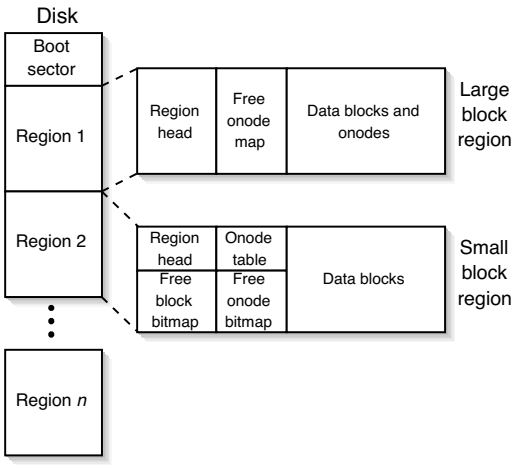


Figure 2. OBFS structure

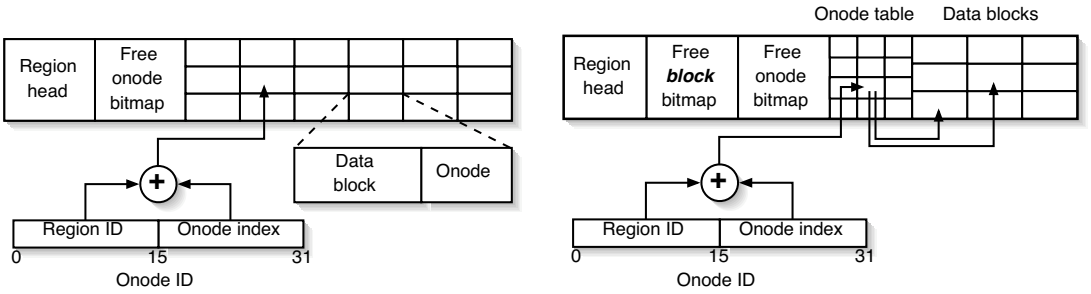
be available. Note that in a small system where an OSD may hold several objects of a file, some locality information may available, but this does not extend to multiple files in the same directory or other tidbits that are useful to general-purpose file systems. Many general-purpose file systems such as Linux Ext2 are extremely inefficient in managing very large directories due to the fact that they do linear search, resulting in $O(n)$ performance on simple directory operations. To avoid this, OBFS uses hash tables (like Ext3 [28]) to organize the objects and achieve much higher performance on directory operations.

5.1. Regions and Variable-Size Blocks

The user-level implementation of OBFS separates the raw disk into regions. As shown in Figure 2, regions are located in fixed positions on the disk and have uniform sizes. All of the blocks in a region have the same size, but the block sizes in different regions may be different. The block size in a free region is undefined until that region is initialized. Regions are initialized when there are insufficient free blocks in any initialized region to satisfy a write request. In this case, OBFS allocates a free region and initializes all of its blocks to the desired block size. When all of the blocks in a used region are freed, OBFS returns the region to the free region list.

Although our region policy supports as many different block sizes as there are regions, too many different block sizes will make space allocation and data management excessively complicated. In our current implementation, OBFS uses two block sizes: small and large. Small blocks are 4 KB, the logical block size in Linux, and large blocks are 512 KB, the system stripe unit size and twice the block size of GPFS (256 KB). Those regions that contain large blocks are called *large block regions* and those regions that contain small blocks are called *small block regions.* With this strategy, large objects can be laid out contiguously on disk in a single large block. The throughput of large objects is greatly improved by the reduction in seek time and reduced metadata operations that are inherent in such a design. Only one disk seek is incurred during the transfer of a large object. OBFS eliminates additional operations on metadata by removing the need for indirect blocks for large objects. Dividing the file system into regions also reduces the size of other FS data structures such as free block lists or maps and thus make the operations on those data structures more efficient.

This scheme reduces file system fragmentation, avoids unnecessary wasted space and more effectively uses the available disk bandwidth. By separating the large blocks in different regions from the



(a) Large Block Region (b) Small Block Region

Figure 3. Region structure and data layout.

small blocks, OBFS can reserve contiguous space for large objects and prevent small objects from using too much space. Region fragmentation will only become a problem in the rare case that the ratio of large to small objects changes significantly during the lifetime of the system, as described in Section 5.6.

Higher throughput in OBFS does not come at the cost of wasted disk space. Internal fragmentation in OBFS is no worse than in a general-purpose Unix file system because the small block size in OBFS is the same as the block size in most Unix file systems. Large blocks do not waste much space because they are only used for objects that will fill or nearly fill the blocks. The only wasted space will be due to objects stored in large blocks that are nearly, but not quite, as large as a stripe unit. This can be limited with a suitable size threshold for selecting the block size to use for an object. One minor complication can occur if an object starts small and then grows past this threshold. Our current implementation recopies the object into a large block when this occurs. Although this sounds expensive, it will happen rarely enough (due to aggressive write coalescing in the client caches) that it does not have a significant impact on system performance, and the inter-region locality of the small blocks makes this a very efficient operation.

5.2. Object Metadata

Object metadata, referred as an *onode*, is used to track the status of each object. Onodes are preallocated in fixed positions at the head of small block regions, similar to the way inodes are placed in cylinder groups in FFS [15]. In large block regions, shown in Figure 3, onodes are packed together with the data block on the disk, similar to embedded inodes [7]. This allows for very low overhead metadata updates as the metadata can be written with the corresponding data block.

Figure 3 shows that each onode has a unique 32-bit identifier consisting of two parts: a 16 bit region identifier and a 16 bit in-region object identifier. If a region occupies 256 MB on disk, this scheme will support OSDs of up to 16 TB, and larger OSDs are possible with larger regions. To locate a desired object, OBFS first finds the region using the region identifier and then uses the in-region object identifier to index the onode. This is particularly effective for large objects because the object index points directly to the onode and the object data, which are stored contiguously.

In the current implementation, onodes for both large and small objects are 512 bytes, allowing OBFS to avoid using indirect blocks entirely. The maximum size of a small object will always be less than the stripe unit size, which is 512 KB in our design. Because the OBFS layout policy assigns objects to a single region, we can use the relative address to track the blocks. Assuming the

region size is 256 MB and the small block size is 4 KB, there will be fewer than 2^{16} small blocks in a region, allowing a two-byte addresses to index all of the blocks in the region. In the worse case, a small object will be a little smaller than 512 KB, requiring 128 data blocks. Thus, the maximum amount of space that may be needed to index the small blocks in an object is 256 bytes, which can easily fit into a 512 byte onode.

5.3. Object Lookup

Given an object identifier, we need to retrieve the object from the disk. In a hierarchical name space, data lookup is implemented by following the path associated with the object to the destination directory and searching (often linearly) for the object in that directory. In a flat name space, linear search is prohibitively expensive, so OBFS uses a hash table, the *Object Lookup Table* (OLT), to manage the mapping between the object identifier and the onode identifier. Each valid object has an entry in the OLT that records the object identifier and the corresponding onode identifier. The size of the OLT is proportional to the number of objects in the OSD: with 20,000 objects residing in an OSD, the OLT requires 233 KB. For efficiency, the OLT is loaded into main memory and updated asynchronously.

Each region has a region head which stores information about the region, including pointers to the free block bitmap and the free onode bitmap. All of the region heads are linked into the Region Head List (RHL). On an 80 GB disk, the RHL occupies 8 MB of disk space. Like the OLT, the RHL is loaded into memory and updated asynchronously. After obtaining an onode identifier, OBFS searches the RHL using the upper 16 bits of the onode identifier to obtain the corresponding region type. If the onode belongs to a large block region, the object data address can be directly calculated. Otherwise, OBFS searches the in-memory onode cache to find that onode. A disk copy of the onode will be loaded into the onode cache if the search fails.

5.4. Disk Layout Policy

The disk layout policy of OBFS is quite simple. For each incoming request, OBFS first decides what type of block(s) the object should use. If the object size is above the utilization threshold of the large blocks, a large block is assigned to the object; otherwise, it uses small blocks.

For those objects that use large blocks, OBFS only needs to find the nearest large-block region that contains a free block, mark it as used, and write the object to that block. For objects that use small blocks, an FFS-like allocation policy is employed. OBFS searches the active region list to find the nearest region that has enough free small blocks for the incoming object. After identifying a region with sufficient space, OBFS tries to find a contiguous chunk of free blocks that is large enough for the incoming object. If such a chunk of blocks is not available, the largest contiguous chunk of blocks in that region is assigned to the object. The amount of space allocated in this step is subtracted from the object size, and the process is repeated until the entire object is stored within the region. If no region has the desired number and type of free blocks, the nearest free region will be initialized and put into the active region list. The incoming object will then be allocated to this new region.

The OBFS data allocation policy guarantees that each of the large objects is allocated contiguously and each of the small objects is allocated in a single region. No extra seeks are needed during a large object transfer and only short seeks are needed to read or write the small objects, no matter how long the file system has been running. Compared with a general-purpose file system, OBFS

is much less fragmented after running for a long time, minimizing performance degradation as the system ages.

5.5. Reliability and Integrity

As mentioned in section 5.3, OBFS asynchronously updates important data structures such as the OLT and the RHL to achieve better performance. In order to guarantee system reliability, OBFS updates some important information in the onodes synchronously. If the system crashes, OBFS will scan all of the onodes on the disk to rebuild the OLT and the RHL. For each object, the object identifier and the region identifier are used to assemble a new entry in the OLT. The block addresses for each object are then used to rebuild each region free block bitmap. Because the onodes are synchronously updated, we can eventually rebuild the OLT and RHL and restore the system. As mentioned above, OBFS updates metadata either without an extra disk seek or with one short disk seek. In so doing, it keeps the file system reliable and maintain system integrity with very little overhead.

5.6. Region Cleaning

Since OBFS uses regions to organize different types of blocks, one potential problem is that there will be no free regions and no free space in regions of the desired type. Unlike LFS [21], which must clean segments on a regular basis, OBFS will *never* need cleaning unless the ratio between large and small objects changes significantly over time on an OSD which has been nearly full. This can only happen when the object size characteristic of the workload changes significantly when the file system is near its capacity. We do not expect this to happen very often in practice. However, if it happens, it can result in many full regions of one type, many underutilized regions of the other type, and no free regions. In this situation, the cleaner can coalesce the data in the underutilized regions and create free regions which can be used for regions of desired type.

If all of the regions are highly utilized, cleaning will not help much: the disk is simply full. Low utilization regions can only be produced when many objects are written to disk and then deleted, leaving "holes" in regions. However, unlike in LFS, these holes are reused for new objects without the need for cleaning. The only time cleaning is needed is when all of the holes are in the wrong kind of region e.g., the holes are in small block regions, and OBFS is trying to write a large block. This situation only occurs when the ratio between large objects and small objects changes. In our experiments, we only observed the need for the cleaner when we artificially changed the workload ratios on a nearly full disk.

Because cleaning is rarely, if ever, necessary, it will have a negligible impact on OBFS performance. However, cleaning can be used to improve file system performance by defragmenting small-block regions to keep blocks of individual objects together. This process would copy all used blocks of a region to a free region on the disk, sorting the blocks as it goes. Because this would occur on a region-by-region basis and because a new region will always have enough free space for all of the blocks in an old region, it would be trivial to implement. The system need never do this, however.

6. OBFS Performance

We compared the performance of OBFS to that of Linux Ext2, Ext3 and XFS. Ext2 is a widely-used general-purpose file system. Ext3 is used by Lustre for object storage and has the same disk layout as Ext2 but adds a journal for reliability. XFS is a modern high-performance general-purpose file

Capacity	80 GB
Controller	Ultra ATA/133
Track-to-track seek	0.8 ms
Average seek	8.5 ms
Rotation speed	7200 RPM
Sustained transfer rate	24.2–44.4 MB/s

Table 1. Specifications of the Maxtor D740X-6L disk used in the experiments

system that uses B-trees and extent-based allocation. While Ext2, Ext3, and XFS run as in-kernel file systems, the version of OBFS used in these experiments is a user-level file system. An in-kernel implementation of OBFS would take advantage of the very effective caching provided by the Linux kernel, but our user-level implementation cannot. Thus, in order to allow for a fair comparison, we executed the following experiments with the system buffer cache bypassed: all of the file systems were mounted using the "-o sync" parameter, which forced the system buffer cache to use a write-through policy. The results generated evaluates disk layout policies of different file systems. With caching enabled, all three file systems will achieve higher performance. We expect the performance change of OBFS to be comparable to those of XFS, Ext2, and Ext3.

6.1. Experimental Setup

All of the experiments were executed on a PC with a 1 GHZ Pentium III CPU and 512 MB of RAM, running Red Hat Linux, kernel version 2.4.0. To examine the performance of the file systems with minimal impact from other operating system activities, we dedicated an 80 GB Maxtor D740X-6L disk (see Table 1) to the experiments. This disk was divided into multiple 8 GB partitions. The first partition was used to install file systems and run experiments. The rest were used to backup aged file system images. We used aged file systems to more accurately measure the long-term performance of the file systems. For each experiment, we copied an aged file system to the first partition of the disk, unmounted the disk and rebooted Linux to clean the buffer cache, then mounted the aged partition to run the benchmarks. We repeated these steps three times and took the average of the performance numbers obtained.

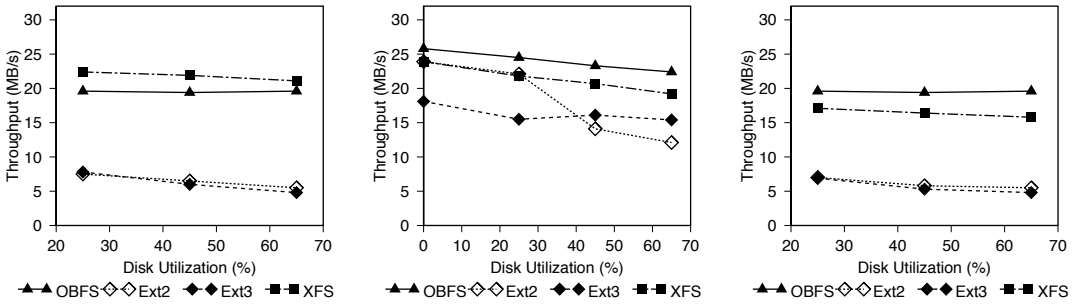
Smith, *et al.* [25] used file system snapshots and traces to approximate the possible activities in file systems. No object-based storage system snapshots are currently available so we used the simplest approach: generate 200,000 to 300,000 randomly distributed create and delete requests and feed these requests to a new file system. The create/delete ratio was dynamically adjusted based on the disk usage, which guaranteed that it neither filled nor emptied the available disk space.

Because our user-level implementation of OBFS bypasses the buffer cache, all three file systems were forced to use synchronous file I/O to allow for a fair comparison of the performance. Ext2 uses asynchronous metadata I/O to achieve high throughput even if synchronous file I/O is used, so we mounted the partitions in synchronous mode to force them to always flush the data in the buffer cache back to disk.

The benchmarks we used consisted of semi-random sequences of object requests whose characteristics were derived from the LLNL workload described in Section 4. On average, 80% of all objects were large objects (512 KB). The rest were small objects whose size was uniformly distributed between 1 KB and 512 KB. To examine the performance of the various file system, we generated two kinds of benchmarks: microbenchmarks and macrobenchmarks. Our microbenchmarks each

	Benchmark I	Benchmark II
	# of ops(total size)	# of ops(total size)
Reads	16854 (7.4GB)	4049 (1.8GB)
Writes	4577 (2.0GB)	8969 (4.0GB)
Rewrites	4214 (1.8GB)	8531 (3.8GB)
Deletes	4356 (1.9GB)	8147 (3.9GB)
Sum	30001 (13.1GB)	29696 (12.5GB)

Table 2. Benchmark parameters



(a) Reads (b) Writes (c) Rewrites

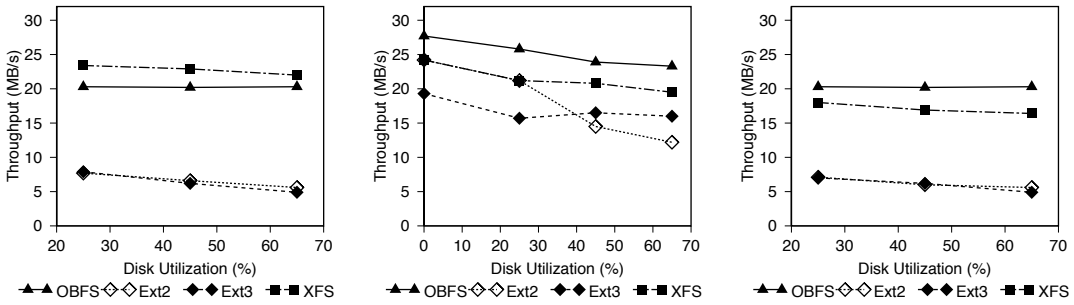
Figure 4. Performance on a workload of mixed-size objects.

consisted of 10,000 requests of a single request type—read, write, or rewrite—and allowed us to examine the performance of the file systems on that request type. Our macrobenchmarks consisted of synthetic workloads composed of create, read, rewrite, and delete operations in ratios determined by the workload mentioned above. These allowed us to examine the performance of the file systems on the expected workload. We used two different macrobenchmarks, Benchmark I and Benchmark II, whose parameters are listed in table 2. Benchmark I is a read-intensive workload in which reads account for 56% of all requests and the total size of the read requests is around 7.4 GB. The writes, rewrites, and deletes account for 15.3%, 14.0%, and 14.5% of the requests. In Benchmark II, reads account for 13.6% of the requests and writes, rewrites, and deletes account for 29.8%, 28.4%, and 27.1%.

6.2. Results

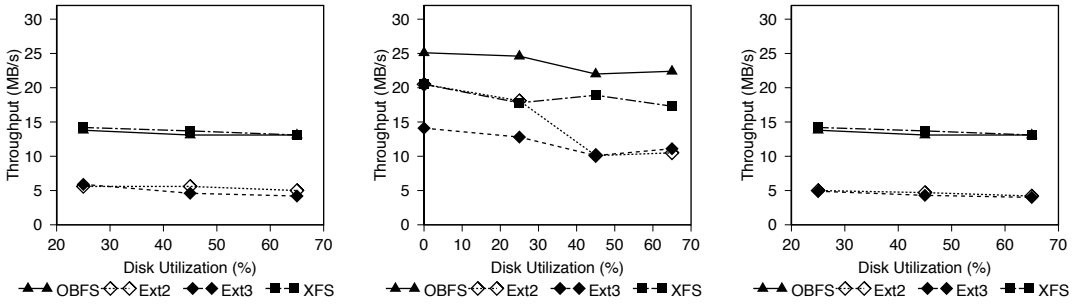
Figure 4 shows the performance of Ext2, Ext3, XFS, and OBFS on a mixed workload consisting of 80% large objects and 20% small objects[1]. As seen in Figure 4(b), OBFS exhibits very good write performance, almost twice that of Ext2 and Ext3 and 10% to 20% better than XFS. The large block scheme of OBFS contributes a lot to the strong write performance. With large blocks, contiguous space has been reserved for the large objects, allowing large objects to be written with only one disk seek. Because OBFS uses regions to organize large and small blocks, limiting the amount of external fragmentation, the performance of OBFS remains good as disk usage increases. At the same time, the performance of Ext2 and Ext3 drops significantly due to the insufficient availability of large contiguous regions, as seen in Figures 4(b), 5(b), and 6(b).

[1]Note that in all of the microbenchmark graphs write performance is displayed starting at 0% disk utilization but because reads and rewrites cannot be done on an empty disk we chose to start those experiments at 25% utilization.



(a) Reads (b) Writes (c) Rewrites

Figure 5. Performance on a workload of large objects.



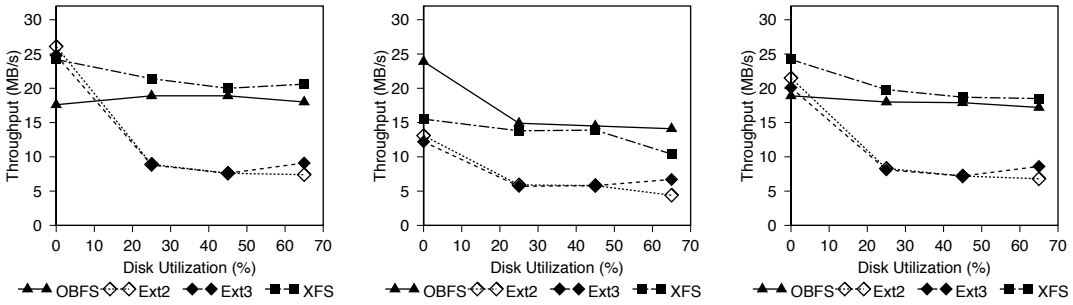
(a) Reads (b) Writes (c) Rewrites

Figure 6. Performance on a workload of small objects.

OBFS outperforms Ext2 and Ext3 by nearly 3 times, but is about 10% slower than XFS, as Figure 4(a) shows. We suspect that a more optimized implementation of XFS contributes to its slightly better read performance. As seen in Figure 4(c), the rewrite performance of OBFS beats that of Ext2 and Ext3 by about 3 times, and beats XFS by about 20–30%. The poor performance of Ext2 and Ext3 in both read and rewrite can be explained by their allocation policies and small blocks. XFS uses extents rather than blocks to organize files, so most files can get contiguous space. This results in excellent performance in both read and write. However, OBFS still shows slightly better performance on object rewrite.

Figure 5 shows the performance of the four file systems on large objects and Figure 6 shows the performance on small objects. Figure 5 is almost the same as Figure 4 because large objects dominate the mixed workload of Figure 4. In Figure 6, we see that OBFS meets the performance of XFS, almost triples the performance of Ext2 and Ext3 when doing reads and rewrites, and exceeds the performance of all three when doing creates.

The benchmark results are shown in Figures 7 and 8. As described above, Benchmark I is a read-intensive workload and Benchmark II is a write-intensive workload. Notice that in our benchmarks, XFS beats both Ext2 and Ext3 by a large margin in all cases; this differs from other benchmark studies [5] that found that Ext2 and XFS have comparable performance. There are three factors in our experiments that favor XFS over Ext2 and Ext3. First, our benchmarks include many large objects, which benefit from XFS extent-based allocation, especially when disk utilization is high. Second, while other benchmarks used fresh disks, our benchmarks use disks subjected to long-term aging [25] to reflect more realistic scenarios. After aging, the performance of Ext2 and Ext3 drops



(a) Reads (b) Writes (c) Overall

Figure 7. Performance under Benchmark I.

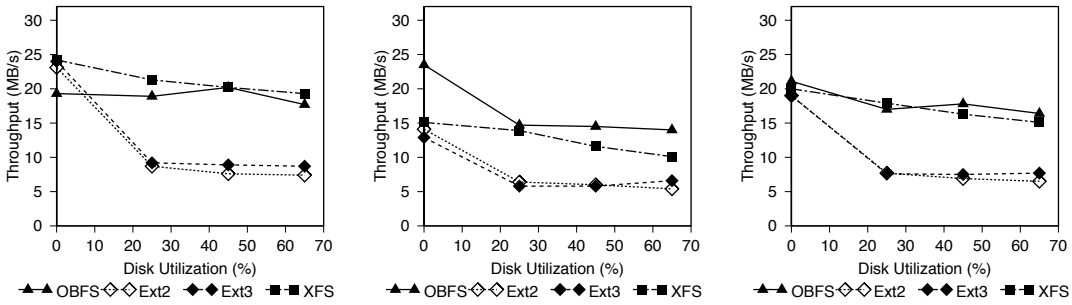
dramatically after aging due to disk fragmentation, while XFS maintains good performance because of its extent-based allocation policy. Third, in our object-based benchmarks, we assume a flat name space in which all objects are allocated in the root directory in all file systems. The linear search of directories used by Ext2 and Ext3 performs poorly when the number of objects scales to tens of thousands. XFS uses B+trees to store its directories, ensuring fast name lookup even in very large directories.

Ext2 and Ext3 outperform OBFS when the disk is nearly empty, as shown in Figures 7(a) and 8(a). This is due in part to the significant likelihood that an object will be in the buffer cache because of the low number of objects that exist in a nearly empty disk. For example, an 8 GB partition at 10% utilization has only 800 MB of data. With 512 MB of main memory, most objects will be in memory and Linux Ext2, Ext3 and XFS reads will proceed at memory speeds while our user-level implementation of OBFS gains no advantage from the buffer cache and must therefore always access the disk. However, as disk usage increases, this effect is minimized and Ext2 and Ext3 read performance decreases rapidly while OBFS performance remains essentially constant. The net result is that OBFS read performance is two or three times that of Ext2 and Ext3. OBFS is still about 10% slower than XFS on reads, similar to the results from earlier read microbenchmarks. OBFS outperforms all three other file systems on writes, however, as Figures 7(b) and 8(b) show. For writes, OBFS is 30–40% faster than XFS and 2–3 times faster than Ext3. Overall, OBFS and XFS are within 10% of each other on the two macrobenchmarks, with one file system winning each benchmark. OBFS clearly beats both Ext2 and Ext3, however, running three times faster on both benchmarks.

Although our macrobenchmarks focused on large-object performance, Figure 6 shows that OBFS meets or exceeds the performance of the other file systems on a workload consisting entirely of small objects, those less than 512 KB. OBFS doubles or triples the performance of Ext2 and Ext3 and matches that of XFS on reads and rewrites and exceeds it by about 25% on writes. As OBFS also does well on large objects, we conclude that it is as well suited to general-purpose object-based storage system workloads as it is to terascale high-performance object-based storage system workloads.

7. Related Work

Many other file systems have been proposed for storing data on disk; however, nearly all of them have been optimized for storing files rather than objects. The Berkeley Fast File System (FFS) [15] and related file systems such as Ext2 and Ext3 [28] are widely used today. They all try to store



(a) Reads (b) Writes (c) Overall

Figure 8. Performance under Benchmark II.

file data contiguously in *cylinder groups*—regions with thousands of contiguous disk blocks. This strategy can lead to fragmentation so techniques such as extents and clustering [16, 24] are used to group blocks together to decrease seek time. Analysis [23, 24] has shown that clustering can improve performance by a factor of two or three, but it is difficult to find contiguous blocks for clustered allocation in aged file systems.

Log-structured file systems [21] group data by optimizing the file system for writes rather than reads, writing data and metadata to segments of the disk as it arrives. This works well if files are written in their entirety, but can suffer on an active file system because files can be interleaved, scattering a file's data among many segments. In addition, log-structured file systems require cleaning, which can reduce overall performance [2].

XFS [19, 27] is a highly optimized file system that uses extents and B-trees to provide high performance. This performance comes at a cost: the file system has grown from 50,000 to nearly 200,000 lines of code, making it potentially less reliable and less attractive for commodity storage devices because such devices cannot afford data corruption due to file system errors. In addition, porting such a file system is a major effort [19].

Gibson, *et al.* have proposed network-attached storage devices [8], but spent little time describing the internal data layout of such devices. WAFL [10], a file system for network-attached storage servers that can write data and metadata to any free location, is optimized for huge numbers of small files distributed over many centrally-controlled disks.

Many scalable storage systems such as GPFS [22], GFS [26], Petal [12], Swift [6], RAMA [17], Slice [1] and Zebra [9] stripe files across individual storage servers. These designs are most similar to the file systems that will use OSDs for data storage; Slice explicitly discusses the use of OSDs to store data [1]. In systems such as GFS, clients manage low-level allocation, making the system less scalable. Systems such as Zebra, Slice, Petal, and RAMA leave allocation to the individual storage servers, reducing the bottlenecks; such file systems can take advantage of our file system running on an OSD. In GPFS, allocation is done in large blocks, allowing the file system to guarantee few disk seeks, but resulting in very low storage utilization for small files.

Existing file systems must do more than allocate data. They must also manage large amounts of metadata and directory information. Most systems do not store data contiguously with metadata, decreasing performance because of the need for multiple writes. Log-structured file systems and embedded inodes [7] store metadata and data contiguously, avoiding this problem, though they still

suffer from the need to update a directory tree correctly. Techniques such as logging [29] and soft updates [14] can reduce the penalty associated with metadata writes, but cannot eliminate it.

8. Conclusions

Object-based storage systems are a promising architecture for large-scale high-performance distributed storage systems. By simplifying and distributing the storage management problem, they provide both performance and scalability. Through standard striping, replication, and parity techniques they can also provide high availability and reliability. However, the workload characteristics observed by OSDs will be quite different from those of general purpose file systems in terms of size distributions, locality of reference, and other characteristics.

To address the needs of such systems, we have developed OBFS, a very small and highly efficient file system targeted specifically for the workloads that will be seen by these object-based storage devices. OBFS currently uses two block sizes: small blocks, roughly equivalent to the blocks in general purpose file systems, and large blocks, equal to the maximum object size. Blocks are laid out in regions that contain both the object data and the onodes for the objects. Free blocks of the appropriate size are allocated sequentially, with no effort made to enforce locality beyond single-region object allocation and the collocation of objects and their onodes.

At present, we have tested OBFS as a user-level file system. Our experiments show that the throughput of OBFS is two to three times that of Linux Ext2 and Ext3, regardless of the object size. OBFS provides slightly lower read performance than Linux XFS, but 10%–40% higher write performance. At a fraction of the size of XFS—2,000 lines of code versus over 50,000 for XFS—OBFS is both smaller and more efficient, making it more suitable for compact embedded implementations. Ultimately, because of its small size and simplicity, we expect that it will also prove to be both more robust and more maintainable than XFS, Ext2, or Ext3.

Finally, we successfully implemented a kernel-level version of the OBFS file system in about three person-weeks. The short implementation time was possible because of OBFS's simplicity and very compact code. At present the performance of our in-kernel implementation does not match that of our user-level implementation because our carefully managed large blocks get broken into small blocks by the Linux buffer management layer, as encountered by the XFS developers. We intend to rewrite the buffer management code, as they did, to avoid this problem. With this change, we expect the in-kernel OBFS performance to exceed that of the user-level implementation, further solidifying OBFS's advantage over general-purpose file systems for use in object-based storage devices.

Acknowledgments

This research was supported by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory under contract B520714. The Storage Systems Research Center is supported in part by gifts from Hewlett Packard, IBM, Intel, LSI Logic, Microsoft, Overland Storage, and Veritas.

References

[1] D. C. Anderson, J. S. Chase, and A. M. Vahdat. Interposed request routing for scalable network storage. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2000.

[2] T. Blackwell, J. Harris, , and M. Seltzer. Heuristic cleaning algorithms in log-structured file systems. In *Proceedings of the Winter 1995 USENIX Technical Conference*, pages 277–288. USENIX, Jan. 1995.

[3] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel*. O'Reilly and Associates, Oct. 2000.

[4] P. J. Braam. The Lustre storage architecture, 2002.

[5] R. Bryant, R. Forester, and J. Hawkes. Filesystem performance and scalability in Linux 2.4.17. In *Proceedings of the Freenix Track: 2002 USENIX Annual Technical Conference*, Monterey, CA, June 2002. USENIX.

[6] L.-F. Cabrera and D. D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *Computing Systems*, 4(4):405–436, 1991.

[7] G. R. Ganger and M. F. Kaashoek. Embedded inodes and explicit groupings: Exploiting disk bandwidth for small files. In *Proceedings of the 1997 USENIX Annual Technical Conference*, pages 1–17. USENIX Association, Jan. 1997.

[8] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 92–103, San Jose, CA, Oct. 1998.

[9] J. H. Hartman and J. K. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3):274–310, 1995.

[10] D. Hitz, J. Lau, and M. Malcom. File system design for an NFS file server appliance. In *Proceedings of the Winter 1994 USENIX Technical Conference*, pages 235–246, San Francisco, CA, Jan. 1994.

[11] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

[12] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 84–92, Cambridge, MA, 1996.

[13] D. Long, S. Brandt, E. Miller, F. Wang, Y. Lin, L. Xue, and Q. Xin. Design and implementation of large scale object-based storage system. Technical Report ucsc-crl-02-35, University of California, Santa Cruz, Nov. 2002.

[14] M. K. McKusick and G. R. Ganger. Soft updates: A technique for eliminating most synchronous writes in the Fast File System. In *Proceedings of the Freenix Track: 1999 USENIX Annual Technical Conference*, pages 1–18, June 1999.

[15] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, Aug. 1984.

[16] L. W. McVoy and S. R. Kleiman. Extent-like performance from a UNIX file system. In *Proceedings of the Winter 1991 USENIX Technical Conference*, pages 33–44. USENIX, Jan. 1991.

[17] E. L. Miller and R. H. Katz. RAMA: An easy-to-use, high-performance parallel file system. *Parallel Computing*, 23(4):419–446, 1997.

[18] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, Mar. 1986.

[19] J. Mostek, B. Earl, S. Levine, S. Lord, R. Cattelan, K. McDonell, T. Kline, B. Gaffey, and R. Ananthanarayanan. Porting the SGI XFS file system to Linux. In *Proceedings of the Freenix Track: 2000 USENIX Annual Technical Conference*, pages 65–76, San Diego, CA, June 2000. USENIX.

[20] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pages 41–54, June 2000.

[21] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, Feb. 1992.

[22] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 231–244. USENIX, Jan. 2002.

[23] M. Seltzer, K. A. Smith, H. Balakrishnan, J. Chang, S. McMains, and V. Padmanabhan. File system logging versus clustering: A performance comparison. In *Proceedings of the Winter 1995 USENIX Technical Conference*, pages 249–264, 1995.

[24] K. A. Smith and M. I. Seltzer. A comparison of FFS disk allocation policies. In *Proceedings of the 1996 USENIX Annual Technical Conference*, pages 15–26, 1996.

[25] K. A. Smith and M. I. Seltzer. File system aging—increasing the relevance of file system benchmarks. In *Proceedings of the 1997 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 203–213, 1997.

[26] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The Global File System. In *Proceedings of the 5th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 319–342, College Park, MD, 1996.

[27] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proceedings of the 1996 USENIX Annual Technical Conference*, pages 1–14, Jan. 1996.

[28] T. Y. Ts'o and S. Tweedie. Planned extensions to the Linux EXT2/EXT3 filesystem. In *Proceedings of the Freenix Track: 2002 USENIX Annual Technical Conference*, pages 235–244, Monterey, CA, June 2002. USENIX.

[29] U. Vahalia, C. G. Gray, and D. Ting. Metadata logging in an NFS server. In *Proceedings of the Winter 1995 USENIX Technical Conference*, New Orleans, LA, Jan. 1995. USENIX.

[30] R. O. Weber. Information technology—SCSI object-based storage device commands (OSD). Technical Council Proposal Document T10/1355-D, Technical Committee T10, Aug. 2002.

Duplicate Data Elimination in a SAN File System

Bo Hong
Univ. of California, Santa Cruz
hongbo@cs.ucsc.edu

Demyn Plantenberg
IBM Almaden Research Center
demyn@almaden.ibm.com

Darrell D.E. Long
Univ. of California, Santa Cruz
darrell@cs.ucsc.edu

Miriam Sivan-Zimet
IBM Almaden Research Center
mzimet@us.ibm.com

Abstract

Duplicate Data Elimination (DDE) is our method for identifying and coalescing identical data blocks in Storage Tank, a SAN file system. On-line file systems pose a unique set of performance and implementation challenges for this feature. Existing techniques, which are used to improve both storage and network utilization, do not satisfy these constraints. Our design employs a combination of content hashing, copy-on-write, and lazy updates to achieve its functional and performance goals. DDE executes primarily as a background process. The design also builds on Storage Tank's FlashCopy function to ease implementation.[1]

We include an analysis of selected real-world data sets that is aimed at demonstrating the space-saving potential of coalescing duplicate data. Our results show that DDE can reduce storage consumption by up to 80% in some application environments. The analysis explores several additional features, such as the impact of varying file block size and the contribution of whole file duplication to the net savings.

1. Introduction

Duplicate data can occupy a substantial portion of a storage system. Often the duplication is intentional: files are copied for safe keeping or for historical records. Just as often, the duplicate data appear through independent channels: individuals who save the same email attachments or who download the same files from the web. It seems intuitive that addressing all of this unrecognized redundancy could result in storage resources being used more efficiently.

Our research goal is to reduce the amount of duplicated data in on-line file systems without significantly impacting system performance. This performance requirement is what differentiates our approach, which we call Duplicate Data Elimination (DDE), from those used in backup and archival storage systems [1, 6, 23]. To minimize its performance impact, DDE executes primarily as a background process that operates in a lazy, best-effort fashion whenever possible. Data is written to the file system as usual, and then some time later, background threads find duplicates and coalesce them to save storage. DDE is transparent to users. It is also flexible enough to be enabled and disabled on an existing file system without disrupting its operation, and flexible enough to be used on parts of the file system, such as select directories or particular file types.

Duplicate Data Elimination (DDE) is designed for IBM Storage Tank [17], a heterogeneous, scalable SAN file system. In Storage Tank, file system clients coordinate their actions through meta-data servers, but access the storage devices directly without involving servers in the data path. DDE uses three key techniques to address its design goals: content-based hashing, copy-on-write (COW), and lazy update. DDE detects duplicate data on the logical block level by comparing hashes of the block contents; it guarantees consistency between block contents and their hashes by using copy-on-write. Data is coalesced by changing corresponding file block allocation maps. COW and lazy update allow us to update the file block allocation maps without revoking the file's data locks. Together these techniques minimize DDE's performance impact.

Figure 1 shows an example of coalescing duplicate data blocks in an on-line file system. Before coalescing, files F_1, F_2 and F_3 consume 11 blocks. However, they each contain a common piece of data that is three blocks in size. Clients are unaware of this duplication when they write these files. The server detects the common data later and coalesces the identical blocks. After coalescing, F_1–F_3 consume only five blocks in total by sharing the same three blocks. Six blocks are saved, resulting in a 55% storage reduction.

[1] Storage Tank technology is available today in the IBM Total Storage SAN File System (SANFS). However, this paper and research is based on underlying Storage Tank technology and may not become part of the IBM TotalStorage SAN File System product.

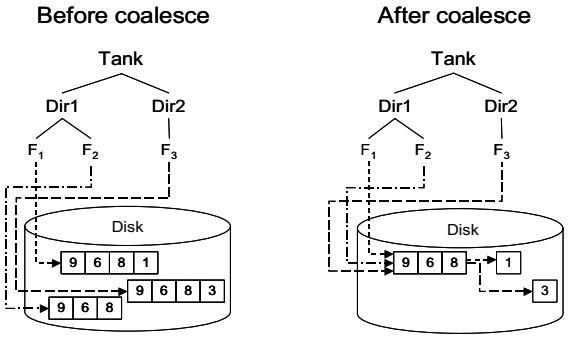


Figure 1. An example of coalescing duplicate data blocks.

2. Background

IBM Storage Tank is a multi-platform, scalable file system that works with storage area networks (SANs) [17]. In Storage Tank, data is stored on devices that can be directly accessed through a SAN, while meta-data is managed separately by one or more specialized Storage Tank meta-data servers. Storage Tank clients are designed to direct all meta-data operations to Storage Tank servers and to direct all data operations to storage devices. Storage Tank servers are not involved in the data path.

The current version of Storage Tank works with ordinary block-addressable storage devices such as disk drives and RAID systems. The basic I/O operation unit in Storage Tank is a block. The storage devices are required to have no more intelligence than the ability to read and write blocks from the volumes (LUNs) they present. Storage Tank file data is also managed in block units. The size of a file block is typically a multiple of the device block size.

Storage Tank exposes three new abstractions called *file sets*, *storage pools*, and *arenas*. These are in addition to the traditional abstractions found in file systems such as files, directories, and volumes. A file set is a subtree of the global namespace. It groups a set of Storage Tank files and directories for the purpose of load balancing and management. A storage pool is a collection of one or more volumes. It provides a logical grouping of the volumes for the allocation of space to file sets. A file set can cross multiple storage pools. An arena provides the mapping between a file set and a storage pool. As such, there is one arena for each file set that has files in a particular storage pool. The arena abstraction is strictly internal to the Storage Tank server, but is an important element in duplicate data elimination. Using an arena, Storage Tank can track the used and free space owned by a file set in a storage pool, and specifies the logical to physical mapping of space in the file set to the volumes in the storage pool.

The Storage Tank Protocol provides a rich locking scheme that enables file sharing among Storage Tank clients or, when necessary, allows clients to have exclusive accesses to files [5, 7, 8]. A Storage Tank server grants locks to clients, and the lock granularity is per file. There are three file lock modes in Storage Tank: 1) exclusive (X), which allows a single client to cache both data and metadata, which it can read and modify; 2) Shared Read (SR), which allows clients to cache data and metadata for read operations, and 3) Shared Write (SW), in which mode clients cannot cache data but can cache metadata in read-only mode. The Storage Tank Protocol also provides copy-on-write capability to support file system snapshots. The server can mark blocks as read-only to enforce copy-on-write.

Storage Tank technology is available today in the IBM Total Storage SAN File System (SANFS). However, this paper and research is based on underlying Storage Tank technology and may not become part of the IBM TotalStorage SAN File System product.

3. Related Work

Data duplication is ubiquitous. Different techniques have been proposed to identify commonality in data, and to exploit this knowledge for reducing storage and network resource consumption due to storing and transferring duplicate data.

Our work was directly inspired by Venti [23]. Venti is a network storage system intended for archival data. In Venti, the unique SHA-1 [12] hash of a block acts as the block identifier, which is used in place of the block address for read and write operations. Venti also implements a write-once policy that prohibits data from being deleted once it is stored. This write-once policy becomes practical, in part, because Venti stores only one copy of each unique data block.

In on-line file systems, performance is essential and data is dynamic and ready to change. This is radically different from the requirements in archival and backup systems, where data is immutable and performance is less of a concern. In our design, duplicate data is also detected on the block level, but in the background. Data is still addressed as usual by the block where it is stored, so data accesses have no extra hash-to-block index searching overheads as in Venti. In turn, it determines that data duplication detection and coalescing is an after-effect effort, *i.e.* it is done after clients have written data blocks to storage devices. The server also maintains a mapping function between block hashes and blocks. A weaker variant of the write-once policy, copy-on-write (COW), is used to guarantee its consistency. Unreferenced blocks due to deletion and COW can be reclaimed.

Single instance store (SIS) [3] also detects duplicate data in an after-effect fashion but on the file level. The technique is optimized for Microsoft Windows remote install servers [18] that store different installation images. In this scenario, the knowledge of file duplication is a priori and files are less likely to be modified. In general on-line file

systems, the granularity of file-level data duplication detection may be too coarse because any modification to a file can cause the loss of the benefit of storage reduction.

LBFS [20] and Pastiche [9] detect data duplication at the granularity of variable-sized chunks, whose boundary regions, called *anchors*, are identified by using the techniques of shingling [16] and Rabin fingerprints [24]. This technique is suitable for backup systems and low-bandwidth network environments, where the reduction of storage and network transmission is more important than performance.

Delta compression is another technique that can effectively reduce duplicate data, thus the requirements of storage and network bandwidth [1, 6, 19, 25]. When a base version of a data object exists subsequent versions can be represented by changes (deltas) to save both storage and network transmission. Delta compression, in general, requires some prior knowledge of data object versioning. It cannot explore common data across multiple files and a change of a (base) file may cause recalculating deltas for other files. In DERD (Delta-Encoding via Resemblance Detection) [11], similar files can be identified as pairs by using data similarity detection techniques [4, 16] without having any specific prior knowledge.

Some file systems provide on-line compression capability [15, 21]. Although it can effectively improve storage efficiency, this technique has significant run-time compression and decompression overheads. On-line compression explores intra-file compressibility and cannot take advantage of common data across files.

The techniques of naming and indexing data objects based on their content hashes are also found in several other systems. In the SFS read-only file system [13], blocks are identified by their SHA-1 hashes and the block hashes are hashed recursively to build up more complex structures. The Stanford digital library repository [10] uses the cyclic redundancy check (CRC) values of data objects as their unique handles. Content-derived names [2, 14] take a similar approach to address the issue of naming and managing reusable software components.

4. Design of Duplicate Data Elimination

Our design goal is to transparently reduce duplicate data in Storage Tank as much as possible without penalizing system performance significantly. Instead of finding data duplication at the first spot, we delay this duplication detection and identify and eliminate duplicate data when server loads are low. In this way, we minimize the performance impact of duplicate data elimination (DDE). In our design, we use three techniques: content-based hashing, copy-on-write (COW), and lazy update.

Duplicate data blocks are detected by the Storage Tank server. A client uses a collision-resistant hash function to digest the block contents it writes to storage devices and returns their hashes to the server. Such a unique hash is called the *fingerprint* of a block. The server compares block fingerprints and coalesces blocks with the same fingerprint (and the same content) by changing corresponding file block allocation maps. The server guarantees consistency between block contents and their fingerprints by directing clients to perform copy-on-write. The server also maintains a reference count for each block and postpones the reclamation of unreferenced blocks. These techniques allow the server to update file block allocation maps without revoking any outstanding data locks on them.

Figure 2 shows the basic idea of duplicate data block elimination in a live file system. The client holds an exclusive (X) lock on file A and a shared-read (SR) lock on file B. These lock modes are described in Section 2. File A and B have the same data stored in blocks 100 and 150 respectively. Before duplicate block coalescing, the server and the client share the same view of files and file block allocation maps. The server finds duplicate data and changes the block allocation map of file B to reference block 100 without updating the client. Even though the client has a stale view on file B, it can still read out the correct data because block 150 is not reclaimed immediately. When the client modifies block 100 of file A, it writes the new content to another block and keeps the content of block 100 intact. Therefore, the content and the fingerprint of block 100 are still consistent and file B still references the right block.

4.1. Duplicate Data Detection

The most straightforward and trusted way of duplicate data detection is bytewise comparison. Unfortunately, it is expensive. An alternative method is to digest data contents by hashing them to much shorter fingerprints and detect duplicate data by comparing their fingerprints. As long as the probability of hash collisions is vanishingly small, we can be confident that two sets of data content are identical if their fingerprints are the same.

In our design, duplicate data is detected on the block level, although maintaining a fingerprint for each block imposes a large amount of bookkeeping information on the system. Storage Tank is based on block-level storage devices, and blocks are the basic operation units. Block-level detection avoids unnecessary I/Os required by other approaches based on files [3] or variable-sized chunks [9, 20], in which the client may have to read other disk blocks before it can recalculate fingerprints due to data boundary mis-alignments. Data duplication detection that is based on blocks has finer granularity and, therefore, higher possibility of storage reduction than techniques based on whole files [3]. Approaches based on variable-sized chunks [9, 20] require at least as much bookkeeping information as the block-based approaches because they tend to limit the average chunk size to obtain a reasonable chance of detecting duplicate data. Chunk-based approaches also need a totally different file block allocation map format from the existing

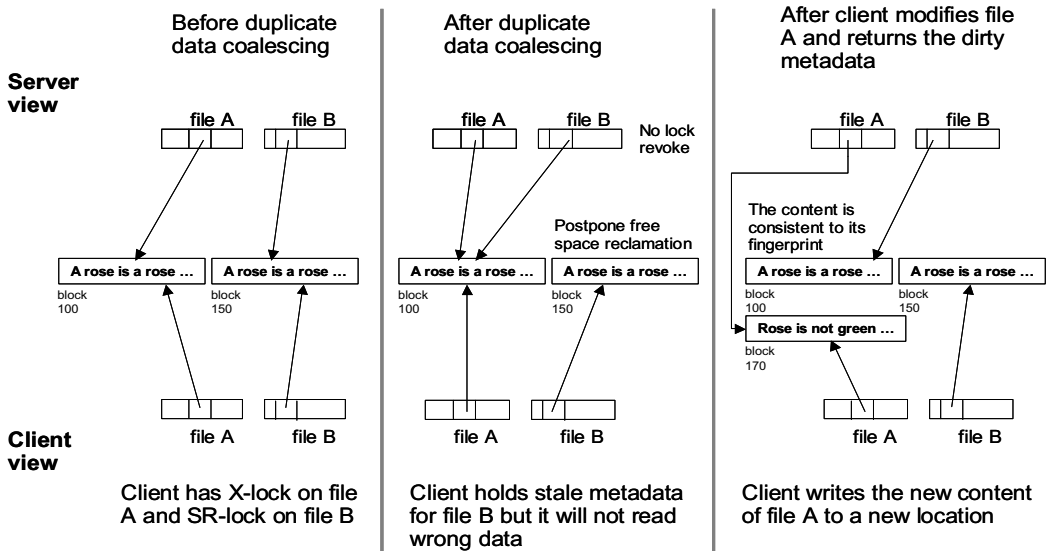


Figure 2. An example of coalescing duplicate data blocks in a live file system.

Storage Tank implementation, which makes them difficult to employ.

DDE uses the SHA-1 hash function [12] to fingerprint block data contents. SHA-1 is a one-way secure hash function with a 160-bit output. Even in a large system that contains an exabyte of data (10^{18} bytes) as 4 kilobyte blocks (roughly 3×10^{14} blocks), the probability of hash collisions using the SHA-1 hash function is less than 10^{-19}, which is at least 5–6 orders of magnitude lower than the probability of an undetectable disk error. To date, there are no known collisions by this hash function. Therefore we can be confident that two blocks are identical if their SHA-1 hashes are the same. In addition, the system could perform bytewise comparisons before coalescing blocks as a cross check.

In Storage Tank, data and meta-data management are separated, and Storage Tank servers are not involved in the data path during normal operations. Disks have little intelligence and cannot detect duplicate data by themselves. Even with smarter disks, without extensive inter-disk communications, each disk would know only its local data fingerprints, which would reduce the chances of detecting duplication. In our design, a client calculates fingerprints of the blocks it writes to storage devices and returns them to the server. Software implementations of SHA-1 are quite efficient and hashing is not a performance bottleneck. Storage Tank servers have a global view of the whole system and are appropriate for data duplication detection.

4.2. Consistency of Data Content and Fingerprints

After we hash the data content of a block, the fingerprint becomes an attribute of the block. Because the fingerprint is eventually stored on the server and the block can be directly accessed and modified by clients, the consistency of the fingerprint and the data content of a block becomes a problem. In Storage Tank, a client can modify a block by two approaches: update-in-place and copy-on-write.

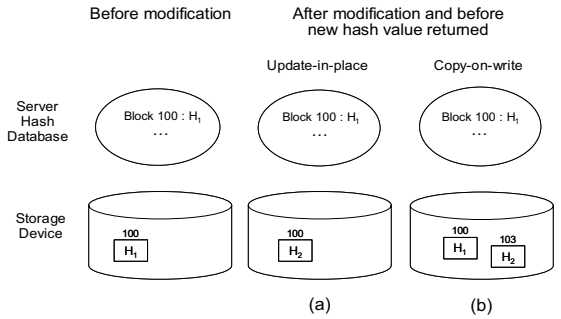


Figure 3. Maintaining consistency between fingerprint and block content under (a) update-in-place and (b) copy-on-write.

4.2.1. Update-in-Place

In Storage Tank, a client can directly modify a block if the block is writable, *i.e.* it writes new data to the same block. This results in inconsistency between the server-side block fingerprint and the block content until the client returns the new fingerprint to the server, as shown in Figure 3(a). The fingerprint of block 100 that the server keeps is inconsistent with the block content until the client returns the latest fingerprint. During the period of inconsistency, any data duplication detection related to this block gives false results.

We can detect this potential inconsistency by checking data locks on the file to which the block belongs. Because

the granularity of file data locks in Storage Tank is per file, the server cannot trust all of the block fingerprints of a file with an exclusive data lock on it.

To avoid erroneous duplication detection and coalescing, we could simply delay DDE on those files with exclusive or shared-write locks. This is feasible in some workloads where only a small fraction of files are active concurrently. However, this approach cannot save any storage in some important environments, such as databases, where applications always hold locks on their files.

Another approach is to revoke data locks on files to force clients to return the latest block fingerprints. This causes two technical problems: lock checking and lock revocation. To check file locks, every block has to maintain a reverse pointer to the file to which it belongs, which makes the bookkeeping information of a block even larger. To guarantee consistency between fingerprints and block contents, every block-coalescing operation has to revoke file locks, if necessary, which can severely penalize the system performance. Therefore, eliminating duplicate data blocks under the update-in-place scenario is inefficient, at best, or impossible.

4.2.2. Copy-on-Write

The basic idea of our work is to eliminate duplicate data blocks by comparing their fingerprints. By using a collision-resistant hash function with a sufficiently large output, such as SHA-1, the fingerprints are considered to be distinct for different data. Therefore, the fingerprint can serve as a unique virtual address for the data content of a block. The mapping function from the virtual address to the physical block address is implicitly provided by the block address itself. Our aim is to make the mapping function nearly one-to-one, *i.e.* each virtual address is mapped to only one physical address.

However, update-in-place violates the basic concept of content-addressed storage by making the mapping function inconsistent. Conceptually, if the content of a block is changed, the new content should be mapped to a new block instead of the original one. Consequently, a client should write modified data to new blocks, which implies a write-once policy, as in Venti [23]. However, write-once keeps all histories of data, which is unnecessary and expensive in on-line file systems. Therefore, we use a weaker variant of write-once, copy-on-write. This technique can guarantee the consistency of the mapping function as long as the original blocks are not reclaimed, as shown in Figure 3(b). The fingerprint of block 100 that the server keeps is still consistent with the block content until block 100 is reclaimed.

Apparently, copy-on-write has a noticeable overhead on normal write operations. Every block modification requires free block allocation because the modified content needs to be written to a new block. However, this cost is less significant than we thought. Some applications, such as Emacs and Microsoft Word, write the whole modified file into a new place, in which case there is no extra cost for COW. The server could also preallocate new blocks to the client that acquires an exclusive or shared-write lock. The most promising approach to alleviate the extra allocation overhead of COW is for the clients to maintain a private storage pool on behalf of the server from which they could allocate locally. Therefore, there is almost no extra cost for COW.

4.3. Lazy Lock Revocation and Free Space Reclamation

The server coalesces duplicate data blocks and reduces storage consumption by updating file block allocation maps to point to one copy of the data and reclaim the rest. During file block allocation map updates, the server does not check whether any client holds data locks on the files. Therefore, the block allocation maps held by the server and clients can be inconsistent, as illustrated in Figure 2. However, we postpone the reclamation of the dereferenced blocks. Therefore, clients holding stale file block allocation maps can still read the correct data from these blocks. At some particular time, *e.g.* midnight, or when the file system is running low on free space, the server revokes all data locks held by clients and frees those dereferenced blocks.

5. Process of Duplicate Data Elimination

Duplicate data elimination is done by the coordination between clients and servers. Simply speaking, clients perform copy-on-write operations and calculate and return block SHA-1 hashes to servers. Servers log clients' activities and identify and coalesce duplicate data blocks in the background. Users are unaware of such operations.

5.1. Impact on Client's Behaviors

In addition to its normal behaviors, a client calculates SHA-1 fingerprints for the data blocks it writes and returns the fingerprints to the server. Because copy-on-write is used (Section 4.2.2), the client does not write modified data blocks back to their original disk blocks; instead, modified data is written to newly-allocated blocks. As long as the client holds a file data lock, further modifications to the same logical block are written to the same newly-allocated disk block. On an update to the server, the client sends the latest block fingerprints along with the block logical offsets within the file and the original physical locations of modified data blocks.

5.2. Data Structures on the Server

We discuss necessary data structure supports on the server that facilitate duplicate data block detection and elimination. Essentially, a reference count table, a fingerprint

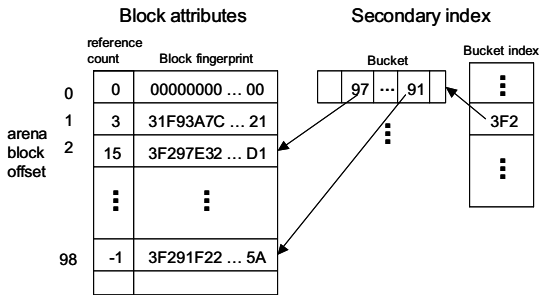


Figure 4. Data structures for storing and retrieving block attributes.

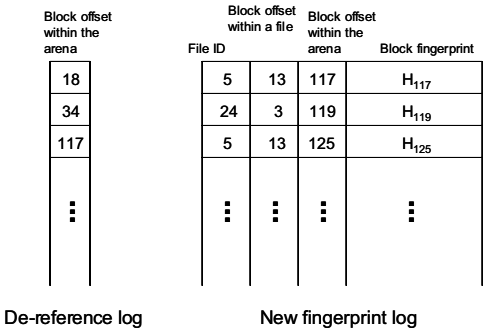


Figure 5. Data structures for logging recent clients' activities.

table and its secondary index maintain attributes associated with blocks, *i.e.* reference counts and fingerprints, as shown in Figure 4; and a dereference log and a new fingerprint log record recent clients' activities, as shown in Figure 5.

The scope of duplicate data elimination is an important design decision. The larger the scope, the higher the degree of data duplication can be, thus providing more benefit. However, for various reasons, people may want to share data only within their working group or their department. Therefore, we limit data duplication detection and elimination within a file set, which essentially is a logical subset of the global namespace. Even within a file set, files can be stored in different storage pools that may belong to different storage classes, which have different characteristics in access latency, reliability, and availability. Sharing data across storage classes can result in noticeable impacts on the quality of storage service. Therefore, we further narrow the scope of DDE within an arena, which provides the mapping between a file set and a storage pool. The data structures we will discuss soon are per arena.

Data are not equally important. Detecting and coalescing temporary, derived, or cached data is less beneficial. Because Storage Tank provides policy-based storage management, a system can be easily configured to store these data in less reliable and so cheaper storage pools, while storing important data in more reliable storage pools. DDE within an arena can take advantage of this flexibility.

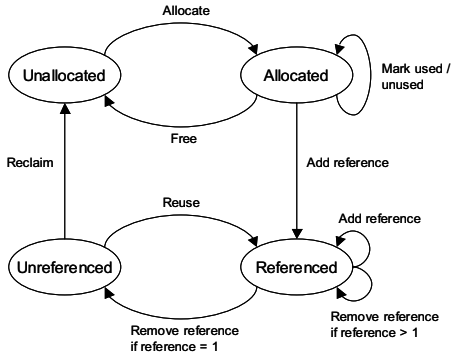


Figure 6. A block is in one of the following states: unallocated, allocated, referenced, and unreferenced.

Because an arena allows the logical to physical mapping of space in the file set to the LUNs in the storage pool, it is equivalent to referencing a block by its physical location and by its logical offset within an arena. For convenience, we will reference an allocated physical block by its logical offset within the arena.

To keep the per-arena data structures to an optimal size, we use 32-bit integers to represent logical block offsets within an arena. Therefore, an arena can contain no more than 2^{32} physical blocks. For 4 KB blocks, an arena can manage 16 TB storage, which is large enough for most applications and environments. However, there is no such a limitation on the capacity of a file set because it can cross multiple storage pools and can consist of multiple arenas.

5.2.1. Reference Count Table

With block coalescing and sharing, a physical block can be referenced multiple times by different files or even one file. Therefore, a reference count is necessary for each block in the arena. From the viewpoint of DDE, a block can be in one of the following four states: 1) free – the block is unallocated; 2) allocated – the block is allocated but unused or it contains valid data that is unhashed; 3) referenced – the block contains valid data, which has been hashed, and is referenced at least once; and 4) unreferenced – the block is allocated and hashed, but has no file referencing it and can be freed or reused. Figure 6 illustrates the four states and the transitions among them.

Without accessing the arena block allocation map, we cannot know whether a block is allocated or not. Fortunately, we are interested in the validity of block fingerprints in our work. Therefore, blocks that are in the first two states (free and allocated) can be merged into one state, invalid, because they contain no valid fingerprints. The reference count of a block indicates its state: 1) invalid, where the reference count is 0, 2) referenced, where the reference count is no less than 1, or 3) unreferenced, where the reference

count is -1. The initial state of a block is invalid because it contains no valid fingerprint until a client calculates it.

A reference count table keeps the reference count for each block in the arena. The table is organized as a linear array, which is indexed by the 32-bit logical block offset within the arena. Each entry in this table is also a 32-bit integer, indicating the state of the corresponding block. The size of the table is up to $2^{32} \times 4$ bytes $= 16$ GB. Because block reference counts are crucial to data integrity, any update on them should be transactional.

A block may contain valid data but has no fingerprint associated with it. Note that the fingerprint of a block is calculated when it is written. If a block is written before the server turns on this feature, it has no fingerprint on the server. A utility running on the server could ask clients to read those data blocks and calculate their fingerprints on behalf of the server.

5.2.2. Fingerprint Table

The fingerprint table keeps unique fingerprints of the blocks in an arena. Each fingerprint in this table is associated with a unique physical block. In other words, the table maintains a one-to-one mapping function between fingerprints and physical blocks. We detect and coalesce duplicate data blocks when we merge the new fingerprint log to this table. The fingerprint table is also organized as a linear array and indexed by the 32-bit logical block offset. Each entry contains a 160-bit SHA-1 fingerprint. A fingerprint is valid only if its block reference count is no less than 1.

The size of the table is up to $2^{32} \times 20$ bytes $= 80$ GB, and it cannot fit in memory. Fortunately, disk block accesses have sequential patterns due to sequential block allocations and file accesses. Therefore, we organize the secure fingerprint table linearly to facilitate comparisons under sequential block accesses. If two disk blocks contain the same content, it is likely that their consecutive blocks also have identical contents. The linear structure makes consecutive fingerprint comparisons efficient because all related entries are in memory.

Conceptually, both the reference count table and the fingerprint table are for describing block attributes and can be merged into one table. Because their sizes are potentially large, and the block reference count is accessed more frequently than the fingerprint, we store them separately to optimize system memory usage.

5.2.3. Secondary Index to Fingerprint Table

Although the linear fingerprint table favors sequential searching, it is difficult to look for a particular fingerprint in this table. Therefore, we also index the table by partial bits of the SHA-1 fingerprint to facilitate random searching. A static hash index is used for this purpose. The hash buckets are indexed by the first 24 bits of the SHA-1 fingerprint. Each bucket contains a 32-bit block pointer. Therefore, the size of the first level index is $2^{24} \times 4$ bytes $= 64$ MB, which can well fit in memory. Each entry in the bucket block contains a 32-bit in-arena logical block offset, indicating the block that the fingerprint is associated with, and the next 32 bits of the SHA-1 fingerprint. Because an arena contains no more than 2^{32} blocks, the average number of hash entries in a bucket is $2^{32}/2^{24} = 2^8 = 256$. When the bucket block size is 4 KB, the average block utilization is $(256 \times (4+4))/4096 = 50\%$. For an arena with capacity much less than 16 TB, multiple buckets can share one bucket block for better storage and memory utilization.

5.2.4. Dereference Log

The dereference log records the in-arena logical offsets of blocks that are recently deleted, dereferenced due to COW by clients, or dereferenced due to block coalescing by the server. We will discuss the third case in greater details in Section 5.3.3. This log is also called *semi-free list* because the blocks in this list could be freed if there is no longer any reference to them. Each entry in this log is a 32-bit integer. To avoid storage leakage, any update on this log should be transactional.

5.2.5. New Fingerprint Log

The new fingerprint log records clients' recent write activities. Each entry in this log includes a 64-bit file ID, a 64-bit logical block offset within the file, a 32-bit logical offset within the arena, and a 160-bit SHA-1 fingerprint. Appending new entries to this log can be non-transactional for the performance purpose because losing the most recent entries only causes losing some opportunities for storage reduction.

5.3. The Responsibilities of the Server

The server enforces a client's copy-on-write behaviors by marking the copy of the file block allocation map in the message buffer as read-only when it responds to a file data lock request from the client. The server immediately logs clients' recent activities, such as delete and write operations. In our design, DDE runs in a best-effort fashion. Therefore, the server lazily detects and coalesces duplicate data blocks and reclaims unused blocks. It also maintains block reference counts and fingerprints correspondingly.

5.3.1. Logging Recent Activities

When the server receives a dirty file block allocation map from a client, it compares it with the one on the server. If a block is marked as unused due to copy-on-write, the server first checks whether it is still referenced by the server-side file block allocation maps. If referenced, the in-arena logical offset of the unused block is appended to the dereference

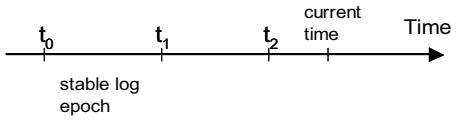


Figure 7. Log epochs.

log; if not (it is possible due to duplicate block coalescing without lock revocation), the block currently referenced by the server-side block allocation map is logged because it is dereferenced by recent modifications on the corresponding file logical block. The unused block returned from the client is not logged because it has been dereferenced due to block coalescing. Other unused blocks are logged.

For each recently-written block, the server also appends an entry to the new fingerprint log, including the identifier of the file to which it belongs, its logical block offset within the file, its logical block offset within the arena, and its fingerprint.

5.3.2. Log Epoch and Preprocessing

We periodically checkpoint the dereference log and the new fingerprint log for two reasons. First of all, the data duplication detection and elimination processes run in a best-effort and background fashion and are unlikely to keep up with the most recent clients' activities. Logging and checkpointing these activities allow the server to detect and coalesce duplicate data blocks during its idle time. By checkpointing the log to epochs, we also limit the number of activities the server processes at a time. Second, recently-written blocks are likely to be modified again. Trying to coalesce these blocks is less beneficial. Therefore, we try to coalesce only blocks in a stable epoch, as shown in Figure 7, whose lifetimes have been long enough.

Assume that we want to merge the new fingerprint log in epoch (t_0, t_1) to the fingerprint table. Because random accesses on the fingerprint table are expensive, we try to reduce the number of fingerprint comparisons by deleting unuseful entries in the new fingerprint log in epoch (t_0, t_1).

First of all, we find overwritten file logical blocks by sorting the log by file ID and logical block offset within a file. We delete the older entries from the log and set their block reference counts to be -1 (unreferenced). We set the block reference counts of other entries in the log to be 1. Second, we scan the dereference log in epoch (t_0, t_1). For each entry in the log, we decrease its block reference count by 1; if the count becomes less than 1, we set it to be -1. Third, we compact the new fingerprint log (t_0, t_1) by deleting those entries also in the dereference logs (t_0, t_1) and (t_1, t_2). The matched entries in the dereference log (t_1, t_2) are removed and their block reference counts are set to be -1.

Note that a fingerprint in the fingerprint table becomes invalid when its block reference count reaches -1. Conceptually, we should also remove its index entry in the secondary index. However, we do not update the secondary index during the log preprocessing because of performance reasons. We postpone the removal of false indexes until the duplicate block coalescing process notices them. False index removal also happens when a secondary index bucket block becomes full.

5.3.3. Merging to Fingerprint Table

We detect and coalesce duplicate data blocks when we merge the compacted new fingerprint log to the fingerprint table. Figure 8 shows the processes of duplication detection and coalescing.

For each entry in the log, we first check whether it has a matching fingerprint in the fingerprint table. If not, we insert the new fingerprint into the table and update the secondary index. If there is an identical fingerprint in the fingerprint table, we check the validity of the fingerprint. If the block reference count of the fingerprint is less than 1, the primary block in the fingerprint table contains no valid data. Therefore, we insert the new fingerprint into the table and update the secondary index. We also need to delete the false index in the secondary index because the previous matching index leads to a block containing invalid data. If the block reference count is no less than 1, we fetch the block allocation map of the file to which the recently-written block belongs, and check whether this block is still referenced by this file. If not, this means that the corresponding logical block in this file was modified after the current fingerprint was returned, and we simply discard this coalescing operation. If the block is still referenced, we update the file block allocation map by referencing the primary block in the fingerprint table without checking or revoking data locks on this file. We also increase the reference count of the primary block and set the reference count of the coalesced block to be -1. When a block is inserted into the fingerprint table, either by adding a new entry or by coalescing to another block, it is marked read-only in the block allocation map of the file to which it belongs.

5.3.4. Free Space Reclamation

A free space reclamation process scans the reference count table in the background. It logs the addresses of the blocks with reference counts -1 and sets their reference counts to 0. At some particular time, *e.g.* midnight, or when the file system is running low on free space, it revokes all data locks and free these blocks.

6. Case Studies

We examined six data sets and studied their degrees of data duplication and their compressibility under common compression techniques. The data sets are summarized in Table 1.

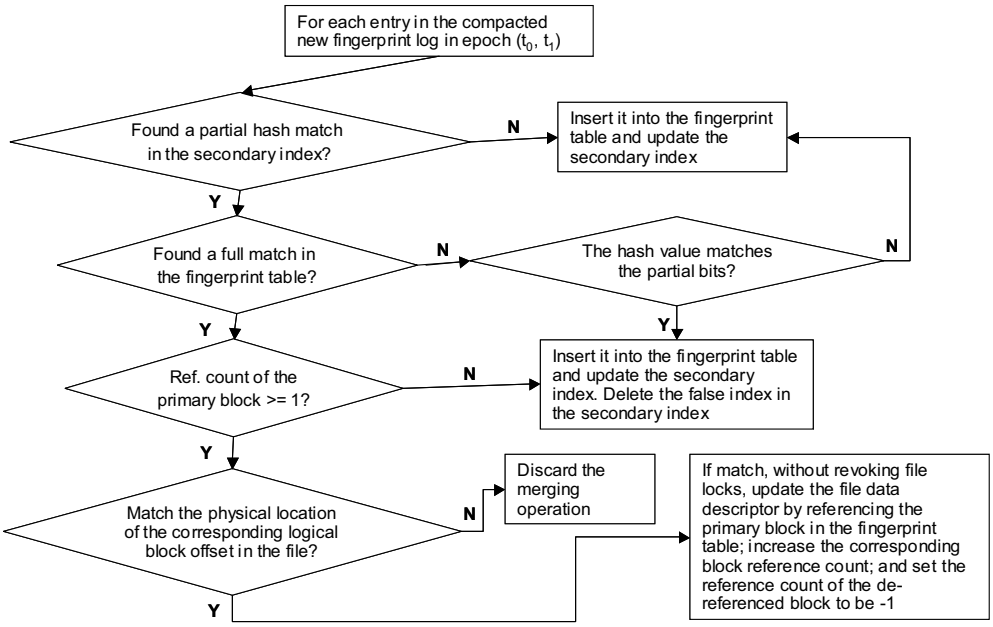


Figure 8. Merge the new fingerprint log to the fingerprint table.

Table 1. Data sets

Name	Description	Size (GB)	Number of files
SNJGSA	file server used by a development team	57	661,729
BVRGSA_BUILD	file server used by the development team for code build	344	2,393,795
BVRGSA_TEST	file server used by the development team for testing	215	115,141
GENOME	human being genome data	348	889,884
LTC_MIRROR	local mirror of installation CDs for different Linux versions	261	241,724
PERSONAL_WORKSTATIONS	aggregation of ten personal workstations	123	879,657

The first three data sets—SNJGSA, BVRGSA_BUILD, and BVRGSA_TEST—are from file servers used by the Storage Tank development team. The servers are used to distribute and exchange data files, but are not used to hold or archive the primary copy of important files. The first server, SNJGSA, is a remote replica that holds a subset of the daily builds that are stored on BVRGSA_BUILD, and a subset of the test data that is on BVRGSA_TEST. In general, the oldest files are deleted when the servers run low on space. The files are almost never overwritten; they tend to be created and then not modified until they are purged few months later.

GENOME contains the human being genome sequence, and is used at UCSC by various bioinformatics applications. The genomic data is encoded in letters, when some single letters and some letter combinations can repeat thousands to millions of times but in fine granularities.

LTC_MIRROR is a local ftp mirror of the IBM Linux Technology Center (LTC) that includes Open Source and IBM software for internal download and use. Among other things, the ftp site holds the CD images (iso) of different Red Hat Linux installations starting at RH7.1 up to RH9.

The last data set is an aggregation of ten personal workstations at the IBM Almaden Research Center that are running Windows. All systems are used for development as well as for general purposes such as email, working documents, etc. We also present the results of these systems when they are analyzed separately.

For each data set, we collected the size and the number of files of the system. We calculated the amount of storage that is required after eliminating data duplication at the granularity of 1 KB blocks. To compare DDE with common compression techniques, we collected the compressed file sizes, using LZO on 64 KB data blocks. LZO (Lempel-Ziv-Oberhumer) is a data compression library that favors speed over compression ratio [22]. We empirically found that the compression capability of LZO is similar to other techniques' when the block size is large. In addition, we calculated the storage reduction achieved by combining the techniques of DDE and LZO. We also calculated what per-

Table 2. DDE and compression results.

Name	% of storage required after DDE (1 KB blocks)	% of storage required after eliminating whole file duplications	% of storage required after LZO on 64 KB blocks	% of storage required after combining DDE and LZO on 64 KB blocks
SNJGSA	32%	55%	56%	30%
BVRGSA_BUILD	21%	62%	67%	35%
BVRGSA_TEST	69%	85%	53%	47%
GENOME	96%	98%	46%	44%
LTC_MIRROR	80%	94%	98%	89%
PERSONAL_WORKSTATIONS	54%	69%	63%	43%

Table 3. DDE and compression results for personal workstations.

System	% of storage required after DDE (1 KB blocks)	% of storage required after eliminating whole file duplications	% of storage required after LZO on 64 KB blocks	% of storage required after combining DDE and LZO on 64 KB blocks
1	66%	71%	61%	43%
2	61%	78%	57%	43%
3	63%	77%	55%	41%
4	77%	91%	63%	55%
5	67%	92%	62%	53%
6	69%	77%	58%	48%
7	70%	80%	61%	49%
8	71%	78%	71%	55%
9	87%	91%	80%	74%
10	73%	84%	57%	48%

centage of the storage is still required after eliminating only whole file duplications.

Table 2 shows the percentage of storage required for different data sets after using DDE at the granularities of 1 KB blocks and whole files, LZO on 64 KB blocks, and the combination of DDE and LZO. DDE at 1 KB blocks only requires one-fifth to one-third of the original storage to hold the BVRGSA_BUILD and SNJGSA data sets and it achieves one to two times higher storage efficiency (the ratio of the amount of logical data to the amount of required physical storage) than LZO at 64 KB blocks. This is because both data sets contain daily builds of the Storage Tank codes that share lots of codes among versions; also the data sets include very small files on average, making LZO inefficient. BVRGSA_TEST on the other hand, has on average larger files (1.9 MB) and the best reduction is achieved when using the combination of LZO compression and DDE on 64 KB blocks. We will study the data set of SNJGSA in greater detail in Section 6.1.

The genomic data set is encoded in letters. Some single letters and letter combinations can repeat thousands to millions of times but at quite fine granularities. With this type of data set it is unlikely to find duplications at the granularities of kilobytes and DDE cannot improve storage efficiency significantly. On the other hand, common compression techniques, such as LZO, are suitable for this data set. The 4% of reduction by DDE is partially due to common file headers and a common "special letter pattern" that fills in gaps in the genomic sequence and partially due to duplicated files that were created by analyzing applications.

Using LZO on the data set of LTC_MIRROR barely reduces storage consumption because generally the files in the Linux installation CD images have already been packaged and compressed. DDE can take advantage of cross-file duplications, mainly from different installation versions, and reduce storage consumption by 20%. A more interesting data set is the actual installation of different Linux versions, instead of the installation images. We plan to look into that in the nearest future.

We also studied the storage requirements of ten personal workstations after using different techniques on individual systems, as shown in Table 3. On average 70% of storage is required for individual systems after applying DDE on them, from which more than half of the saving is due to eliminating whole file duplications. LZO compression can provide better storage efficiency on these systems. By combining both LZO and DDE on 64 KB blocks, the percentage of storage required for individual systems is further reduced to 51% on average. When aggregating files from all machines together, which is potentially what happens with enterprise-level backup applications, the aggregated storage requirement of ten personal workstations by using DDE drops significantly, from 70% to 54%, as shown as PERSONAL_WORKSTATIONS in Table 2. This is because the more individual systems are aggregated, the higher the degree of data duplication is likely to be. Consequently, DDE can result in tremendous storage savings for back up systems that potentially backup hundreds to thousands of personal workstations on a daily basis, which shows a very good example of an application that could benefit from data duplication elimination at the file system. Combining DDE

and LZO on 64 KB blocks can further reduce the storage requirement to 43%.

The granularity of duplicate data detection affects the effectiveness of DDE. Table 2 shows that file-level detection can lose up to 50–70% of the chance of finding duplicate data at 1 KB blocks. We also noticed that using both DDE and LZO on 64 KB blocks does not always generate better results than using DDE solely. In general, smaller block sizes favor DDE because of finer granularities on duplication elimination; larger block sizes favor LZO because of more efficient encoding. In fact, DDE and LZO-like compression techniques are orthogonal when they are operated at the same data unit granularity because compression explores intra-unit compressibility and DDE explores inter-unit duplications.

6.1. Detailed Study on SNJGSA

Figure 9 shows the results of applying DDE to the SNJGSA data set using a range of file system block sizes. The results are given as a percentage of the data blocks that are unique. The block size varies from 512 bytes to 64 KB. As expected, the smallest block size works best because it enables the detection of shorter segments of duplicate data. Interestingly, DDE's effectiveness starts to improve again when the block size reaches 32 KB. The reason is that the file system is "wasting" space using these larger blocks, and DDE is proportionally coalescing more of this wasted space. This effect begins to outpace the reduced number of duplicate blocks due to coarser block granularities. Note that Storage Tank does not support blocks sizes smaller than 4 KB; these results are included to show the savings we are missing out on. The additional space saving potential of smaller blocks is modest: 5%, for instance, between 1 KB and 4 KB blocks in the SNJGSA1 data set.

SNJGSA's usage pattern and its "FIFO" style space management allow us to use this data to simulate a growing file system. Figure 10 shows the amount of space saved by DDE as the file system, which starts empty, grows to eventually contain all the files in the SNJGSA data set. The files are added in order of their modified times (mtime). 4 KB blocks are used. The rightmost value on the chart is 38%, which matches the space savings shown in Figure 9.

If Figure 10 had shown a gradual improvement starting at 100% (no compression) and monotonically decreasing to 38%, we could assert that duplication in this data set is independent of time. This is the type of curve that would be generated if the files were inserted in random order rather than being sorted by mtime. If the chart had shown a flat curve, this would indicate that data duplication is highly correlated in time, *i.e.* has strong temporal locality. The duplicate blocks found in a new file would likely match blocks that were added, for instance, within the last 24 hours, but very unlikely to match blocks that were added a month ago. DDE performs most efficiently on highly time correlated duplication because duplicate blocks tend to be coalesced within a log epoch and require fewer updates to the fingerprint table. At the same time, uncorrelated duplication allows DDE to produce a continually improving compression ratio as the data set grows.

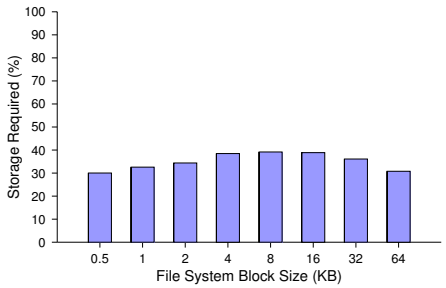


Figure 9. SNJGSA, percentage of storage required after removing duplicate data.

The SNJGSA data set consists of 15 million 4 KB blocks, among which 3.3 million are referenced only once. Among the remaining 11.7 million blocks, only 2.5 million are unique. Figure 11 shows the cumulative distributions of storage savings made by frequently occurring data blocks. It reveals that only 1% of unique blocks contributes to 29% of the total storage savings. This suggests us that even a small amount of hash cache on the client side could explore the frequency of data duplication and save actual I/O on the first spot. Although we have not had the opportunity to study the recency of data duplication, we believe that with careful design, the client-side hash cache can also take advantage of this recency to improve system performance. Furthermore, the spatial and temporal localities of duplicate data generation are dominant factor of DDE's performance on the server side.

SNJGSA and BVRGSA_BUILD demonstrate applications that can substantially benefit from DDE. The engineers using these file servers exploit storage space to make their jobs easier. One way they do this is by creating hundreds of "views" of their data in isolated directory trees, each with a dedicated purpose. However, these pragmatic, technology savvy users do not use a snap-shot facility to create these views, or a configuration management package, or even symbolic links. The engineers employ the most robust and familiar mechanisms available to them: copying the data, applying small alterations, and leave the results to linger indefinitely. In this environment, the file system is the data management application, and DDE extends its reach.

7. Discussions

Often duplicate data copies are made for the purpose of reliability, preservation, and performance. Typically, such duplicate copies are and should be stored in different stor-

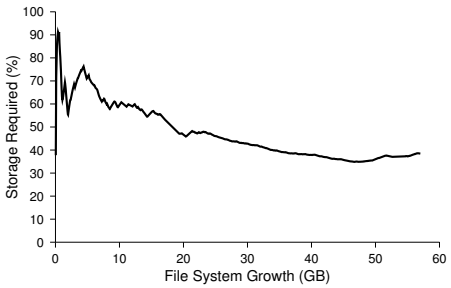


Figure 10. SNJGSA, percentage of unique blocks in a simulated growing file system.

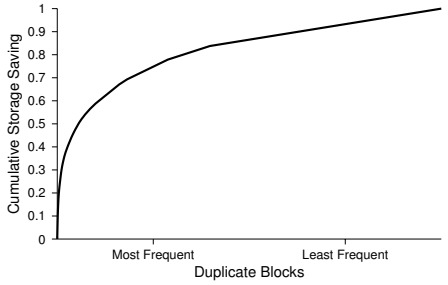


Figure 11. SNJGSA, cumulative contribution to the total storage savings made by coalescing frequently occurring data blocks. Blocks are sorted by their reference frequencies.

age pools. Since the technique of duplicate data elimination is scoped to an arena, data on different storage pools will not be coalesced and its reliability, preservation, and the performance of accessing data will not be affected. If the duplicate data copies are made against inadvertent deletion or corruption of file data, DDE allows copies to be made without consuming additional storage space and still protects against deletion or corruption. However, DDE does create an exposure to potential multiple file corruptions due to a single block error.

DDE coalesces duplicate data blocks in files. It can result in file fragmentation thus degrade the system performance due to non-contiguous reads. This can be alleviated by only coalescing duplicate data with sizes of at least N contiguous blocks. In Section 6, we also found that the majority of duplicate blocks come from whole files, in which cases reading those files has no additional seek overheads. It is probable that DDE can reduce system write activities if clients can detect duplicate data before writing to storage devices, which will be discussed further in Section 8. DDE can also potentially improve the storage subsystem cache utilization because only unique data blocks will be cached.

The degree of data duplication can vary dramatically in different systems and application environments. DDE is a technique that is suitable for those environments with expected high degrees of data duplication such as backup of multiple personal workstations, and it is not necessarily intended for general uses.

The key techniques used in DDE are content-based hashing, copy-on-write, and lazy updates. With necessary and appropriate supports, these techniques thus DDE could be also applicable to other file systems besides Storage Tank. Particularly, lazy updates minimize the performance impact of identifying duplicate data and maintaining block metadata, which will also be beneficial to other file systems.

8. Future Work

We are working on implementing duplicate data elimination in Storage Tank. Besides implementation, there are several research directions we can explore in the future.

The technique of copy-on-write plays a key role in our design to guarantee consistency between fingerprints and block contents. However, it forces a client to request new block allocations for each file modification and has noticeable overheads on normal write operations. To alleviate the extra allocation cost due to COW, the server could preallocate new blocks to a client that acquires an exclusive or shared-write lock. Therefore, the preallocation policy for COW is an interesting research topic. Another promising approach to alleviate this overhead is to allow a client to maintain a small private storage pool on behalf of the server. Therefore, there is almost no extra cost for COW.

In our current design, we employ quite a naive coalescing policy: when we find a recently-written block containing the same data as a block in the fingerprint table, we simply dereference the new block and change the corresponding file block pointer to the primary block in the fingerprint table. This policy is suboptimal in terms of efficiency. Although we have considered file and block access patterns in our design, we do not, for simplicity, explicitly elaborate policies that favor sequential fingerprint probing and matching under certain block access patterns. Therefore, further research on such policies is needed.

The naive coalescing policy we describe in this paper may also result in file fragmentation. Coalescing few blocks within a large file is less desirable. Therefore, a study on policies for minimizing file fragmentation is interesting. Furthermore, a good coalescing policy could reduce storage fragmentation by reusing the lingering unused blocks due to lazy free space reclamation.

The fingerprint of a block is a short version of its data. A client can easily keep a history of its recent write activities by maintaining a fingerprint cache. The client can do part of the duplicate data elimination work in conjunction with the server. More beneficially, actual write operations to storage can be saved if there are cache hits.

As far as we know, there are no extensive and intensive studies on the duplicate data distributions of the block level or other levels. A better understanding of data duplication in

file systems can be enormously beneficial for making good duplicate data coalescing policies.

In our design, we add two attributes on physical disk blocks: the reference count and the SHA-1 fingerprint of the block content. We also provide appropriate data structures to store and retrieve these attributes. Our work makes it feasible to check data integrity in Storage Tank. A client can ensure that the data it reads is the data it writes by comparing the fingerprint on the server with the one calculated from the data it recently reads.

9. Conclusions

Although disk prices drop dramatically, storage is still a precious resource in computer systems. For some data sets, reducing storage consumption caused by duplicate data can significantly improve storage usage efficiency. By using techniques of content-based hashing, copy-on-write, lazy lock revocation, and lazy free space reclamation, we can detect and coalesce duplicate data blocks in on-line file systems without a significant impact on system performance. Our case studies show that 20–79% of storage can be saved by the technique of duplicate data elimination at 1 KB blocks in some application environments. File-level duplication detection is sensitive to changes of file contents and can lose up to 50–70% of the chance of finding duplicate data at finer granularities.

Acknowledgments

We are grateful to Robert Rees, Wayne Hineman, and David Pease for their discussions and insights in Storage Tank. We thank Scott Brandt, Ethan Miller, Feng Wang, and Lan Xue of the University of California at Santa Cruz for their helpful comments on our work. We thank Vijay Sundaram of the University of Massachusetts at Amherst, in particular, for his invaluable discussions. We also thank Terrence Furey and Patrick Gavin of the bioinformatics department of UCSC for providing access to the GENOME data set and helping us to understand the results. Our shepherd Curtis Anderson provided useful feedback and comments on our first draft.

References

[1] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer. Compactly encoding unstructured inputs with differential compression. *Journal of the Association for Computing Machinery*, 49(3):318–367, May 2002.

[2] K. Akala, E. Miller, and J. Hollingsworth. Using content-derived names for package management in Tcl. In *Proceedings of the 6th Annual Tcl/Tk Conference*, pages 171–179, San Diego, CA, Sept. 1998. USENIX.

[3] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in Windows 2000. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 13–24. USENIX, Aug. 2000.

[4] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences (SEQUENCES '97)*, pages 21–29. IEEE Computer Society, 1998.

[5] R. Burns. *Data Management in a Distributed File System for Storage Area Networks*. Ph.d. dissertation, Department of Computer Science, University of California, Santa Cruz, Mar. 2000.

[6] R. Burns and D. D. E. Long. Efficient distributed back-up with delta compression. In *Proceedings of I/O in Parallel and Distributed Systems (IOPADS '97)*, pages 27–36, San Jose, Nov. 1997. ACM.

[7] R. Burns, R. Rees, and D. D. E. Long. Safe caching in a distributed file system for network attached storage. In *Proceedings of the 14th International Parallel & Distributed Processing Symposium (IPDPS 2000)*. IEEE, May 2000.

[8] R. Burns, R. Rees, L. J. Stockmeyer, and D. D. E. Long. Scalable session locking for a distributed file system. *Cluster Computing Journal*, 4(4), 2001.

[9] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 285–298, Boston, MA, Dec. 2002.

[10] A. Crespo and H. Garcia-Molina. Archival storage for digital libraries. In *Proceedings of the Third ACM International Conference on Digital Libraries (DL '98)*, pages 69–78, Pittsburgh, Pennsylvania, June 1998. ACM.

[11] F. Douglis and A. Iyengar. Application-specific delta-encoding via resemblance detection. In *Proceedings of the 2003 USENIX Annual Technical Conference*, pages 113–126. USENIX, June 2003.

[12] Secure hash standard. FIPS 180-1, National Institute of Standards and Technology, Apr. 1995.

[13] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 181–196, San Diego, CA, Oct. 2000.

[14] J. Hollingsworth and E. Miller. Using content-derived names for configuration management. In *Proceedings of the 1997 Symposium on Software Reusability (SSR '97)*, pages 104–109, Boston, MA, May 1997. IEEE.

[15] S. Kan. ShaoLin CogoFS – high-performance and reliable stackable compression file system. http://www.shaolinmicro.com/product/cogofs/, Nov. 2002.

[16] U. Manber. Finding similar files in a large file system. Technical Report TR93-33, Department of Computer Science, The University of Arizona, Tucson, Arizona, Oct. 1993.

[17] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM Storage Tank—a heterogeneous scalable SAN file system. *IBM Systems Journal*, 42(2):250–267, 2003.

[18] Microsoft Windows 2000 Server online help file. Microsoft Corporation, Feb. 2000.

[19] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of SIGCOMM97*, pages 181–194, 1997.

[20] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 174–187, Oct. 2001.

[21] R. Nagar. *Windows NT File System Internals: A Developer's Guide*. O'Reilly and Associates, 1997.

[22] M. F. Oberhumer. oberhumer.com: LZO data compression library. http://www.oberhumer.com/opensource/lzo/, July 2002.

[23] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In D. D. E. Long, editor, *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 89–101, Monterey, California, USA, 2002. USENIX.

[24] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[25] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '00)*, pages 87–95, Stockholm, Sweden, Aug. 2000. ACM Press.

Clotho: Transparent Data Versioning at the Block I/O Level

Michail D. Flouris
Department of Computer Science,
University of Toronto,
10 King's College Road,
Toronto, Ontario M5S 3G4, Canada
Tel: +1-416-978-6610, Fax: +1-416-978-1931
e-mail: flouris@cs.toronto.edu

Angelos Bilas[1]
Institute of Computer Science
Foundation for Research and Technology - Hellas
Vassilika Vouton, P.O.Box 1385,
GR 711 10 Heraklion, Greece
Tel: +30-281-039-1600, Fax: +30-281-039-1601
e-mail: bilas@ics.forth.gr

Abstract

Recently storage management has emerged as one of the main problems in building cost effective storage infrastructures. One of the issues that contribute to management complexity of storage systems is maintaining previous versions of data. Up till now such functionality has been implemented by high-level applications or at the filesystem level. However, many modern systems aim at higher scalability and do not employ such management entities as filesystems.

In this paper we propose pushing the versioning functionality closer to the disk by taking advantage of modern, block-level storage devices. We present *Clotho*, a storage block abstraction layer that allows transparent and automatic data versioning at the block level. *Clotho* provides a set of mechanisms that can be used to build flexible higher-level version management policies that range from keeping all data modifications to version capturing triggered by timers or other system events.

Overall, we find that our approach is promising in offloading significant management overhead and complexity from higher system layers to the disk itself and is a concrete step towards building self-managed storage devices. Our specific contributions are: (i) We implement *Clotho* as a new layer in the block I/O hierarchy in Linux and demonstrate that versioning can be performed at the block level in a transparent manner. (ii) We investigate the impact on I/O path performance overhead using both microbenchmarks as well as SPEC SFS V3.0 over two real filesystems, Ext2FS and ReiserFS. (iii) We examine techniques that reduce the memory and disk space required for metadata information.

1. Introduction

Storage is currently emerging as one of the major problems in building tomorrow's computing infrastructure. Future systems will provide tremendous storage, CPU processing, and network transfer capacity in a cost-efficient manner and they will be able to process and store ever increasing amounts of data. The cost of managing these large amounts of stored data becomes the dominant complexity and cost factor for building, using, and operating modern storage systems. Recent studies [10] show that storage expenditures represent more than 50% of the typical server purchase price for applications such as OLTP (On-Line Transaction Processing) or ERP (Enterprise Resource Planning) and these percentages will keep growing. Furthermore, the cost of storage administration is estimated at several times the purchase price of the storage hardware [2, 5, 7, 12, 33, 34, 36]. Thus, building self-managed storage devices that reduce management-related overheads and complexity is of paramount importance.

One of the most cumbersome management tasks that requires human intervention is creating, maintaining, and recovering previous versions of data for archival, durability, and other reasons. The problem is exacerbated as the capacity and scale of storage systems increases. Today, backup is the main mechanism used to serve these needs. However, traditional backup systems are limited in the functionality they provide. Moreover they usually incur high access and restore overheads on magnetic tapes, they impose a very coarse granularity in the allowable archival periods, usually at least one day, and they result in significant management overheads [5, 27]. Automatic versioning, in conjunction with increasing disk capacities, has been proposed [5, 27] as a method to address these issues. In particular, magnetic disks are becoming cheaper and larger and it is projected that disk storage will soon be as competitive as tape storage [5, 9]. With the advent of inexpensive high-capacity disks, we can perform continuous, real-time versioning and we can maintain online repositories of archived data. Moreover, *online storage versioning* offers a new range of possibilities compared to simply recovering users' files that are available today only in expen-

[1] Also, with the Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, GR 714 09, Greece

sive, high-end storage systems:

- **Recovery from user mistakes.** The users themselves can recover accidentally deleted or modified data by rolling-back to a saved version.
- **Recovery from system corruption.** In the event of a malicious incident on a system, administrators can quickly identify corrupted data as well as recover to a previous, consistent system state [28, 30].
- **Historical analysis of data modifications.** When it is necessary to understand how a piece of data has reached a certain state, versioning proves a valuable tool.

Our goal in this paper is to provide online storage versioning capabilities in commodity storage systems, in a *transparent* and *cost-effective* manner. Storage versioning has been previously proposed and examined purely at the filesystem level [24, 26] or at the block level [14, 31] but being filesystem aware. These approaches to versioning were intended for large, centralized storage servers or appliances. We argue that to build self-managed storage systems, versioning functionality should be pushed to lower system layers, closer to the disk to offload higher system layers [30]. This is made possible by underlying technologies that drive storage systems. Disk storage, network bandwidth, processor speed, and main memory are reaching speeds and capacities that make it possible to build cost-effective storage systems with significant processing capabilities [9, 11, 13, 22] that will be able to both store vast amounts of information [13, 17] and to provide advanced functionality.

Our approach of providing online storage versioning is to provide all related functionality at the block level. This approach has a number of advantages compared to other approaches that try to provide the same features either at the application or the filesystem level. First, it provides a higher level of transparency and in particular is completely filesystem agnostic. For instance, we have used our versioned volumes with multiple, third party, filesystems without the need for any modifications. Data snapshots can be taken on demand and previous versions can be accessed online simultaneously with the current version. Second, it reduces complexity in higher layers of storage systems, namely the filesystem and storage management applications [15]. Third, it takes advantage of the increased processing capabilities and memory sizes of active storage nodes and offloads expensive host-processing overheads to the disk subsystem, thus, increasing the scalability of a storage archival system [15].

However, block-level versioning poses certain challenges as well: (i) Memory and disk space overhead: Because we only have access to blocks of information, depending on application data access patterns, there is increased danger for higher space overhead in storing previous versions of data and the related metadata. (ii) I/O path performance overhead: It is not clear at what cost versioning functionality can be provided at the block-level. (iii) Consistency of the versioned data when the versioned volume is used in conjunction with a filesystem. (iv) Versioning granularity: Since versioning occurs at a lower system layer, information about the content of data is not available, as is, for instance, the case when versioning is implemented in the filesystem or the application level. Thus, we only have access to full volumes as opposed to individual files.

We design *Clotho*[2], a system that provides versioning at the block-level and addresses all above issues, demonstrating that this can be done at minimal space and performance overheads. First, *Clotho* has low memory space overhead and uses a novel method to avoid copy-on-write costs when the versioning extent size is larger than the block size. Furthermore, *Clotho* employs off-line *differential compression* (or diffing) to reduce disk space overhead for archived versions. Second, using advanced disk management algorithms, *Clotho*'s operation is reduced in all cases to simply manipulating pointers in in-memory data structures. Thus, *Clotho*'s common-path overhead follows the rapidly increasing processor-memory curve and does not depend on the much lower disk speeds. Third, *Clotho* deals with version consistency by providing mechanisms that can be used by higher system layers to guarantee that either all data is consistent or to mark which data (files) are not. Finally, we believe that volumes are an appropriate granularity for versioning policies. Given the amounts of information that will need to be managed in the future, specifying volume-wide policies and placing files on volumes with the appropriate properties, will result in more efficient data management.

We implement *Clotho* as an additional layer (driver) in the I/O hierarchy of Linux. Our implementation approach allows *Clotho* the flexibility to be inserted in many different points in the block layer hierarchy in a single machine, a clustered I/O system, or a SAN. *Clotho* works over simple block devices such as a standard disk driver or more advanced device drivers such as volume managers or hardware/software RAIDs. Furthermore, our implementation provides to higher layers the abstraction of a standard block device and thus, can be used by other disk drivers, volume/storage managers, object stores or filesystems.

We evaluate our implementation with both microbenchmarks as well as real filesystems and the SPEC SFS 3.0 suite over NFS. The main memory overhead of *Clotho* for metadata is about 500 Kbytes per GByte of disk space and can be further reduced by using larger extents. Moreover, the performance overhead of *Clotho* for I/O operations is

[2]Clotho, one of the Fates in ancient Greek mythology, spins the thread of life for every mortal.

minimal, however, it may change the behavior of higher layers (including the filesystem), especially if they make implicit assumptions about the underlying block device, e.g. the location of disk blocks. In such cases, co-design of the two layers, or system tuning maybe necessary to not degrade system performance. Overall, we find that our approach is promising in offloading significant management overhead and complexity from higher system layers to the disk itself and is a concrete step towards building self-managed storage systems.

The rest of this paper is organized as follows. Section 2. presents our design and discusses the related challenges in building block-level versioning systems. Section 3. presents our implementation. Section 4. presents our experimental evaluation and results. Section 5. discusses related work, while section 6. presents limitations and future work. Finally, Section 7. draws our conclusions.

2. System Design

The design of *Clotho* is driven by the following high-level goals and challenges:

- Flexibility and transparency.
- Low metadata footprint and low disk space overhead.
- Low-overhead common I/O path operation.
- Consistent online snapshots.

Next we discuss how we address each of these challenges separately.

2.1. Flexibility and Transparency

Clotho provides versioned volumes to higher system layers. These volumes look similar to ordinary physical disks that can, however, be customized, based on user-defined policies to keep previous versions of the data they store. Essentially, *Clotho* provides a set of mechanisms that allow the user to add time as a dimension in managing data by creating and manipulating volume versions. Every piece of data passing through *Clotho* is indexed based not only on its location on the block device, but also on the time the block was written. When a new version is created, a subsequent write to a block will create a new block preserving the previous version. Multiple writes to the same data block between versions result in overwriting the same block. Using *Clotho*, device versions can be captured either on demand or automatically at prespecified periods. The user can view and access all previous versions of the data online, as independent block devices along with the current version. The user can compact and/or delete previous volume versions. In this work we focus on the mechanisms *Clotho* provides and we only present simple policies we have implemented and tested ourselves. We expect that systems administrators will further define their own policies in the context of higher-level storage management tools.

Clotho provides a set of primitives (mechanisms) that higher-level policies can use for automatic version management:

- `CreateVersion()` provides a mechanism for capturing the lower-level block device's state into a version.
- `DeleteVersion()` explicitly removes a previously archived version and reclaims the corresponding volume space.
- `ListVersions()` shows all saved version of a specific block device.
- `ViewVersion()` enables creating a virtual device that corresponds to a specific version of the volume and is accessible in read-only mode.
- `CompactVersion()` and `UncompactVersion()` provide the ability to compact and uncompact existing versions for reducing disk space overhead.

Versions of a volume have the following properties: Each version is identified by a unique *version number*, which is an integer counter starting from value 0 and increasing with each new version. Version numbers are associated with timestamps for presentation purposes. All blocks of the device that are accessible to higher layers during a period of time will be part of the version of the volume taken at that moment (if any) and will be identified by the same version number. Each of the archived versions exists solely in read-only state and will be presented to the higher levels of the block I/O hierarchy as a distinct, virtual, read-only block device. The latest version of a device is both readable and writable, exists through the entire lifetime of the *Clotho*'s operation, and cannot be deleted.

Clotho can be inserted arbitrarily in a system's layered block I/O hierarchy. This stackable driver concept has been employed to design other block-level I/O abstractions, such as software RAID systems or volume managers, in a clean and flexible manner [31]. The *input* (higher) layer can be any filesystem or other block-level abstraction or application, such as a RAID, volume manager, or another storage system. *Clotho* accepts block I/O requests (read, write, ioctl) from this layer. Similarly, the *output* (lower) layer can be any other block device or block-level abstraction. This design provides great flexibility in configuring a system's block device hierarchy. Figure 1 shows some possible configurations for *Clotho*. On the left part of Figure 1, *Clotho* operates on top of a physical disk device. In the middle, *Clotho* acts as a wrapper of

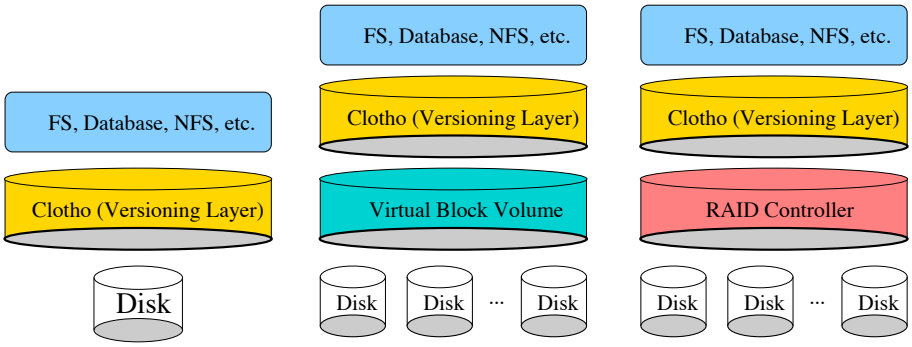


Figure 1: *Clotho* in the block device hierarchy.

a single virtual volume constructed by a volume manager, which abstracts multiple physical disks. In this configuration *Clotho* captures versions of the whole virtual volume. On the right side of Figure 1, *Clotho* is layered on top of a RAID controller which adds reliability to the system. The result is a storage volume that is both versioned and can tolerate disk failures.

Most higher level abstractions that are built on top of existing block devices assume a device of fixed size, with few rare exceptions such as resizable filesystems. However, the space taken by previous versions of data in *Clotho*, depends on the number and the amount of modified data between them. *Clotho* can provide both a fixed size block device abstraction to higher layers, as well as dynamically resizable devices, if the higher layers support it. At device initialization time *Clotho* reserves a configurable percentage of the available device space for keeping previous versions of the data. This essentially partitions (logically not physically) the capacity of the wrapped device into two logical segments as illustrated in Figure 2. The *Primary Data Segment* (PDS), which contains the data of the current (latest) version and the *Backup Data Segment* (BDS), which contains all the data of the archived versions. When BDS becomes full, *Clotho* simply returns an appropriate error code and the user has to reclaim parts of the BDS by deleting or compacting previous versions, or by moving them to some other device. These operations can also be performed automatically by a module that implements high-level data management policies. The latest version of the block device continues to be available and usable at all times. *Clotho* enforces this capacity segmentation by reporting as its total size to the input layer, only the size of the PDS. The space reserved for storing versions is hidden from the input layer and is accessed and managed only through the API provided by *Clotho*.

Finally, *Clotho*'s metadata needs to be saved on the output device along with the actual data. Losing metadata used for indexing extents would render the data stored throughout the block I/O hierarchy unusable. This is similar to most block-level abstractions, such as volume managers, and software RAID devices. *Clotho* stores metadata to the output device periodically. The size of the metadata depends on the size of the encapsulated device and the extent size. In general, *Clotho*'s metadata are much less than the metadata of a typical filesystem and thus, saving them to stable storage is not an issue.

2.2. Reducing Metadata Footprint

The three main types of metadata in *Clotho* are the *Logical Extent Table* (LXT), the *Device Version List* (DVL), and the *Device Superblock* (DSB).

The *Logical Extent Table (LXT)* is a structure used for logical to physical block translation. *Clotho* presents to the input layer *logical* block numbers as opposed to the *physical* block numbers provided by the wrapped device. Note that these block numbers need not directly correspond to actual physical locations, if another block I/O abstraction, such as a volume manager (e.g. LVM [32]) is used as the output layer. *Clotho* uses the LXT to translate logical block numbers to physical block numbers.

The *Device Version List (DVL)* is a list of all versions of the output device that are available to higher layers as separate block devices. For every existing version, it stores its version number, the virtual device it may be linked to, the version creation timestamp, and a number of flags.

The *Device Superblock (DSB)* is a small table containing important attributes of the output versioned device. It stores information about the capacity of the input and output device, the space partitioning, the size of the extents, the sector and block size, the current version counter, the number of existing versions and other usage counters.

The LXT is the most demanding type of metadata and is conceptually an array indexed by block numbers. The basic block size for most block devices varies between 512 Bytes (the size of a disk sector) and 8 KBytes. This results

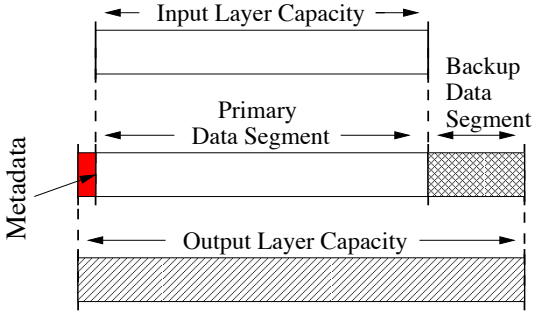


Figure 2: Logical space segments in *Clotho*.

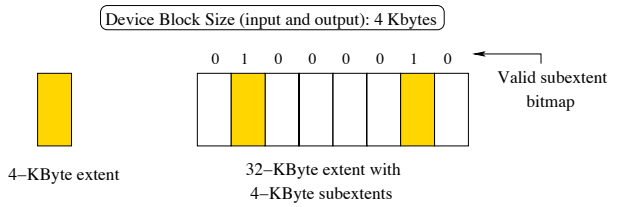


Figure 3: Subextent addressing in large extents.

in large memory requirements. For instance, for 1 TByte of disk storage with 4-KByte blocks, the LXT has 256M entries. In the current version of *Clotho*, every LXT entry is 128-bits (16 bytes). These include 32 bits for block addressing and 32 bits for versions that allow for a practically unlimited number of versions. Thus, the LXT requires about 4 GBytes per TByte of disk storage. Note that a 32-bit address space, with 4 KByte blocks, can address 16 TBytes of storage.

To reduce the footprint of the LXT and at the same time increase the addressing range of LXT, we use *extents* as opposed to device blocks as our basic data unit. An extent is a set of consecutive (logical and physical) blocks. Extents can be thought as *Clotho*'s internal block size, which one can configure to arbitrary sizes, up to several hundred KBytes or a few MBytes. Similarly to physical and logical blocks, we denote extents as logical (input) extents or physical (output) extents. We have implemented and tested extent sizes ranging from 1 KByte to 64 KBytes. With 32-KByte extents and subextent addressing, we need only 500 MBytes of memory per TByte of storage. Moreover with a 32-KByte extent size we can address 128 TBytes of storage.

However, large extent sizes may result in significant performance overhead. When the extent size and the operating system block size for *Clotho* block devices are the same (e.g. 4KBytes), *Clotho* receives from the operating system the full extent for which it has to create a full version. When using extents larger than this maximum size, *Clotho* sees only a subset of the extent for which it needs to create a new version. Thus, it needs to copy the rest of the extent in the new version, even though only a small portion of it written by the higher system layers. This copy can significantly decrease performance in the common I/O path, especially for large extent sizes. However, large extents are desirable for reducing metadata footprint. Given that operating systems support I/O blocks of up to a maximum size (e.g. 4K in Linux), this may result in severe performance overheads.

To address this problem we use *subextent addressing*. Using a small (24-bit) bitmap in each LXT entry we need not copy the whole extent in a partial update. Instead we just translate the block write to a subextent of the same size and mark it in the subextent bitmap as valid, using just 1 bit. In a subsequent read operation we search for the valid subextents in the LXT before translating the read operation. For a 32-Kbyte extent size, we need only 8 bits in the bitmap for 4-KByte subextents.

Another possible approach to reduce memory footprint is to store only part of the metadata in main memory and perform swapping of active metadata from stable storage. However, this solution is not adequate for storage systems where large amounts of data need to be addressed. Moreover it is orthogonal to subextent addressing and can be combined with it.

2.3. Version Management Overhead

All version management operations can be performed at a negligible cost by manipulating in-memory data structures. Creating a new version in *Clotho* involves simply incrementing the current version counter and does not involve copying any data. When `CreateVersion()` is called, *Clotho* stalls all incoming I/O requests for the time required to flush all its outstanding writes to the output layer. When everything is synchronized on stable storage, *Clotho* increases the current version counter, appends a new entry to the device version list, and creates a new virtual block device that can be used to access the captured version of the output device, as explained later. Since each version is linked to exactly one virtual device, the (OS-specific) device number that sends the I/O request can be used to retrieve the I/O request's version.

The fact that device versioning is a low-overhead operation makes it possible to create flexible versioning policies. Versions can be created by external processes periodically or based on system events. For instance, the user processes can specify that it requires a new version every 1 hour, or whenever all files to the device are closed or on every single write to the device. Some of the mechanisms to detect such events, e.g. if there are any open files on a device, may be (and currently are) implemented in *Clotho*

but could also be provided by other system components.

In order to free backup disk space, *Clotho* provides a mechanism to delete volume versions. On a `DeleteVersion()` operation, *Clotho* traverses the primary LXT segment and for every entry that has a version number equal to the delete candidate, changes the version number to the next existing version number. It then traverses the backup LXT segment and frees the related physical extents. As with version creation, all operations for version deletion are performed in-memory and can overlap with regular I/O. `DeleteVersion()` is provided to higher layer in order to implement *version cleaning policies*. Since storage space is finite, such policies are necessary in order to continue versioning without running out of backup storage. Finally, even if the backup data segment (BDS) is full, I/O to the primary data segment and the latest version of data can continue without interruption.

2.4. Common I/O Path Overhead

We consider the common path for *Clotho*, as the I/O path to read and write to the latest (current) version of the output block device, while versioning occurs frequently. Accesses to older versions are of less importance since they are not expected to occur as frequently as current version usage. Accordingly, we divide read and write access to volume versions in two categories, accesses to the current version and accesses to any previous version. The main technique to reduce common path overhead is to divide the LXT in two logical segments, corresponding to the primary and backup data segments of the output device as illustrated in Figure 2. The primary segment of the LXT (mentioned as PLX in figures) has an equal number of logical extents as the input layer to allow a direct, 1-1 mapping between the logical extents and the physical extents of the current version on the output device. By using a direct, 1-1 mapping, *Clotho* can locate a physical extent of the *current version* of a data block with a *single lookup* in the primary LXT segment, when translating I/O requests to the current version of the versioned device. If the input device needs to access previous versions of a versioned output device, then multiple accesses to the LXT maybe required to locate the appropriate version of the requested extent.

To find the physical extent that holds the specific version of the requested block, *Clotho* first references the primary LXT segment entry to locate the current version of the requested extent (a single table access). Then it uses the linked list that represents the version history of the extent to locate the appropriate version of the requested block. Depending on the type of each I/O request and the state of the requested block, I/O requests can be categorized as follows:

Write requests can only be performed on the current version of a device, since older versions are read-only. Thus, *Clotho* can locate the LXT entry of a current version extent with a *single LXT access*. Write requests can be one of three kinds as shown in Figure 4:

a. Writes to new, unallocated blocks. In this case, *Clotho* calls its extent allocator module, which returns an available physical extent of the output device, it updates the corresponding entry in the LXT, and forwards the write operation to the output device. The *extent allocation policy* in our current implementation is a scan-type policy, starting from the beginning of the PDS to its end. Free extents are ignored until we reach the end of the device, when we rewind the allocation pointer and start allocating the free extents.

b. Writes to existing blocks that have been modified after the last snapshot was captured (i.e. their version number is equal to the current version number). In this case *Clotho* locates the corresponding entry in the primary LXT segment with a single lookup and translates the request's block address to the existing physical block number of the output device. Note that in this case the blocks are *updated in place*.

c. Writes to existing blocks that have not been modified since the last snapshot was captured (i.e. their version number is lower than the current version number). The data in the existing physical extent must not be overwritten, but instead the new data should be written in a different location and a new version of the extent must be created. *Clotho* allocates a new LXT entry *in the backup segment* and *swaps* the old and new LXT entries so that the old one is moved to the backup LXT segment. The block address is then translated to the new physical extent address, and the request is forwarded to the output layer. This "swapping" of LXT entries maintains the 1-1 mapping of current version logical extents in the LXT which optimizes common-path references to a single LXT lookup.

This write translation algorithm allows for independent, fine grain versioning at the extent level. Every extent in the LXT is versioned according to its updates from the input level. Extents that are updated more often have more versions than extents written less frequently.

Read request translation is illustrated in Figure 5. First *Clotho* determines the desired version of the device by the virtual device name and number in the request (e.g. /dev/clt1-01 corresponds to version 1 and /dev/clt1-02 to version 2). Then, *Clotho* traverses the version list on the LXT for the specific extent or subextent and locates the appropriate physical block.

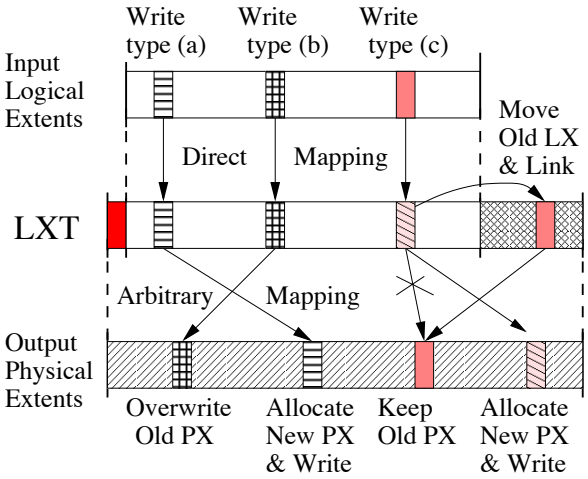


Figure 4: Translation path for write requests.

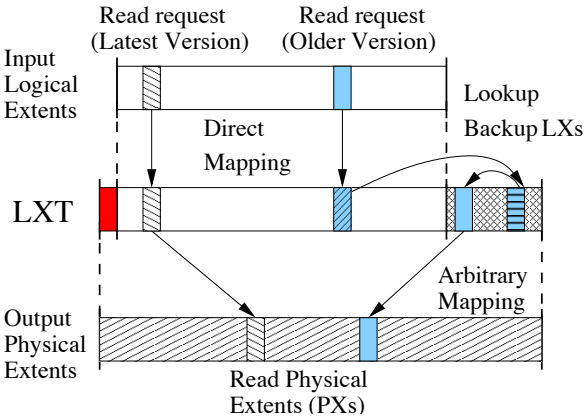


Figure 5: Translation path for read requests.

Finally, previous versions of a *Clotho* device appear as different virtual block devices. Higher layers, e.g. filesystems, can use these devices to access old versions of the data. If the device id of a read request is different from the normal input layer's device id, the read request refers to an extent belonging to a previous version. *Clotho* determines from the device id the version of the extent requested. Then, it traverses the version list associated with this extent to locate the backup LXT entry that holds the appropriate version of the logical extent. This translation process is illustrated in Figure 5.

2.5. Reducing Disk Space Requirements

Since *Clotho* operates at the block level, there is an induced overhead in the amount of space it needs to store data updates. If an application for instance, using a file modifies a few consecutive bytes in the file, *Clotho* will create a new version for the full block that contains the modified data. To reduce the space overhead in *Clotho* we provide a differential, content-based compaction mechanism, which we describe next.

Clotho provides the user with the ability to compact device versions and still be able to transparently access them online. The policy decision on when to compact a version is left to higher-layers in the system, similarly to all policy decisions in *Clotho*. We use a form of binary differential compression [1] to only store the data that has been modified since the last version capture. When `CompactVersion()` is called, *Clotho* constructs a differential encoding (or delta) between the blocks that belong to a given version with corresponding blocks in its previous version [1]. Although a lot of differential policies can be applied in this case, such as to compare the content of a specific version with its next version, or both the previous and the next version, at this stage we only explore diffing with the previous version. Furthermore, although versions can also be compressed independently of differential compression using algorithms such as Lempel-Ziv encoding [37] or Wheeler-Burrows encoding [3], this is beyond the scope of our work. We envision that such functionality can be provided by other layers in the I/O device stack.

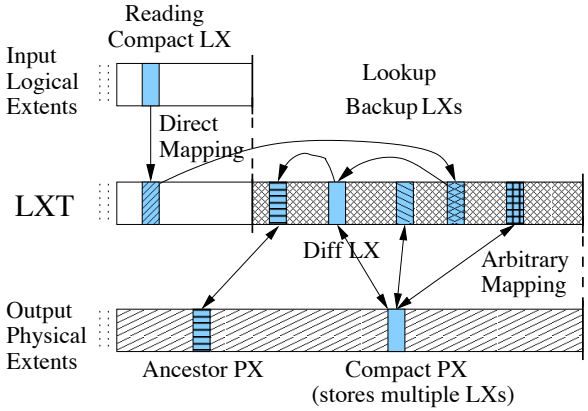


Figure 6: Read translation for compact versions.

The differential encoding algorithm works as follows. When a compaction operation is triggered, the algorithm runs through the backup data segment of the LXT and locates the extents that belong to the version under consideration. If an extent does not have a previous version, it is not compacted. For each of the extents to be compacted the algorithm locates its previous version, diffs the two extents, and writes the diffs to a physical extent on the output device. If the diff size is greater than a threshold and diffing is not very effective, then *Clotho* discards this pair and proceeds with the next extent of the version to be compacted. In other words, *Clotho*'s differential compression algorithm works selectively on the physical extents, compacting only the extents that can be reduced in size. The

rest are left in their normal format to avoid performance penalties necessary for their reconstruction.

Since the compacted form of an extent requires less size than a whole physical extent, the algorithm stores multiple deltas in the same physical extent, effectively imposing a different structure on the output block device. Furthermore, for compacted versions, multiple entries in the LXT may point to the same physical extent. The related entries in the LXT and the ancestor extent are kept in *Clotho*'s metadata. Physical extents that are freed after compaction are reused for storage. Figure 6 shows sample LXT mappings for a compacted version of the output layer.

Data on a compacted version can be accessed transparently online as data on uncompacted volumes (Figure 6). *Clotho* follows the same path to locate the appropriate version of the logical extent in the LXT. To recreate the original, full extent data we need the differential data of the previous version of the logical extent. With this information *Clotho* can reconstruct the requested block and return it to the input driver. We evaluate the related overheads in Section 4..

Clotho supports recursive compaction of devices. The next version of a compacted version can still be compacted. Also, compacted versions can be uncompacted to their original state with the reverse process. A side-effect of the differential encoding concept is that it creates dependences between two consecutive versions of a logical extent, which affects the way versions are accessed, as explained next.

When deleting versions, *Clotho* checks for dependencies of compacted versions on a previous version and does not delete extents that are required for un-diffing, even if their versions are deleted. These logical extents are marked as `"shadow"` and are attached to the compacted version. It is left to higher-level policies to decide if keeping such blocks increases the space overhead and it would be better to uncompact the related version and delete any shadow logical extents.

2.6. Consistency

One of the main issues in block device versioning at arbitrary times is consistency of the stored data. There are three levels of consistency for online versioning:

System state consistency: This refers to consistency of system buffers and data structures that are used in the I/O path. To deal with this, *Clotho* flushes all device buffers in the kernel as well as filesystem metadata before version creation. This guarantees that the data and metadata on the block device correspond to a valid snapshot of the filesystem at a point-in-time. That is, there are no consistency issues in internal system data structures.

Open file consistency: When a filesystem is used on top of a versioned device, certain files may be open at the time of a snapshot. Although *Clotho* does not deal with this issue, it provides a mechanism to assist users. When a new version is created, *Clotho*'s user-level module queries the system for open files on the particular device. If such files are open, *Clotho* creates a special directory with links to all open files and includes the directory in the archived version. Thus, when accessing older versions the user can find out which files were open at versioning time.

Application consistency: Applications using the versioned volume may have a specialized notion of consistency. For instance, an application may be using two files that are both updated atomically. If a version is created after the first file is updated and closed but before the second one is open and updated, then, although no files were open during version creation, the application data may still be inconsistent. This type of consistency is not possible to deal with transparently without application knowledge or support, and thus, is not addressed by *Clotho*.

3. System Implementation

We have implemented *Clotho* as a block device driver module in the Linux 2.4 kernel and a user-level control utility, in about 6,000 lines of C code. The kernel module can be loaded at runtime and configured for any output layer device by means of an `ioctl()` command triggered by the user-level agent. After configuring the output layer device, the user can manipulate the *Clotho* block device depending on the higher layer that they want to use. For instance, the user can build a filesystem on top of a *Clotho* device with `mkfs` or `newfs` and then `mount` it as a regular filesystem.

Our module adheres to the framework of block I/O devices in the Linux kernel and provides two interfaces to user programs: an `ioctl` command interface and a `/proc` interface for device information and statistics. All operations described in the design section to create, delete, and manage version have been implemented through the `ioctl` interface and are initiated by the user-level agent. The `/proc` interface provides information about each device version through readable ASCII files. *Clotho* also uses this interface to report a number of statistics, including the times of creation, a version's time span, the size of modified data from the previous version and some specific information to compacted versions, such as the compaction level and the number of *shadow* extents.

The *Clotho* module uses the zero-copy mechanism of the `make_request_fn()` fashion that is used by LVM [32]. With this mechanism *Clotho* is able to translate the device driver id `(kdev_t)` and the sector address of a block re-

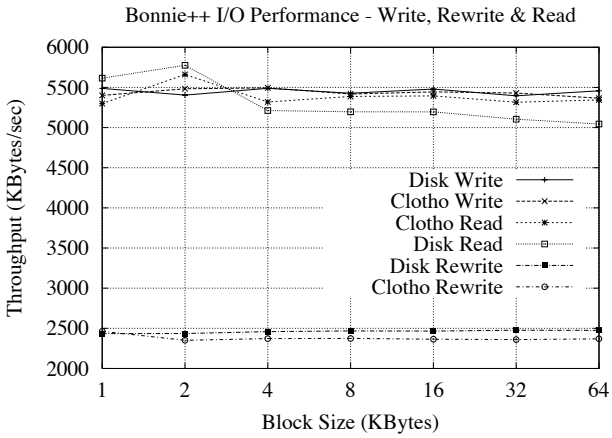


Figure 7: Bonnie++ throughput for write, rewrite, and read operations.

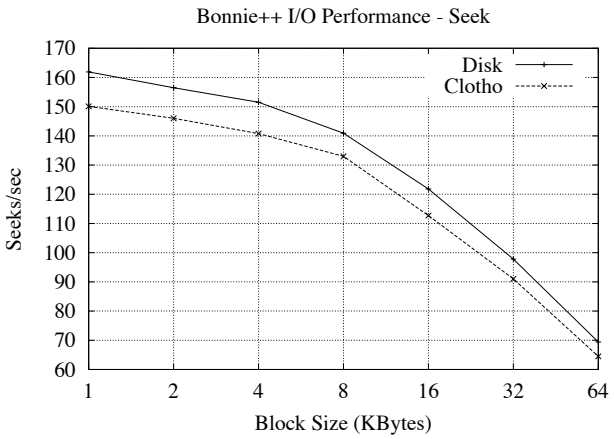


Figure 8: Bonnie "seek and read" performance.

quest (`struct buffer_head`) and redirect it to other devices with minimal overhead. To achieve persistence of metadata, *Clotho* uses a kernel thread created at module load time, which flushes the metadata to the output layer at configurable (currently 30s) intervals.

The virtual device creation uses the partitionable block device concepts in the Linux kernel. A limit in the Linux kernel minor numbering is that there can be at most 255 minor numbers for a specific device and thus, only 255 versions can be seen simultaneously as partitions of *Clotho*. However, the number of partitions supported by *Clotho* can be much larger. To overcome this limitation we have created a mechanism through an `ioctl` call that allows the user to link and unlink on demand any of the available versions to any of the 255 minor number partitions of a *Clotho* device. As mentioned, each of these partitions is read-only and can be used as a normal block device, e.g. can be mounted to a mount-point.

4. Experimental Results

Our experimental environment consists of two identical PCs running Linux. Each system has two Pentium III 866 MHz CPUs, 768 MBytes of RAM, an IBM-DTLA-307045 ATA Hard Disk Drive with a capacity of 46116 MBytes (2-MByte cache), and a 100MBps Ethernet NIC. The operating system is Red Hat Linux 7.1, with the 2.4.18 SMP kernel. All experiments are performed on a 21-GByte partition of the IBM disk. With a 32-KByte extent, we need only 10.5 MBytes of memory for our 21-GByte partition.

Although a number of system parameters are worth investigation, we evaluate *Clotho* with respect to two parameters: memory and performance overhead. We use two extent sizes, 4 and 32 KBytes. Smaller extent sizes have higher memory requirements. For our 21-GByte partition, *Clotho* with 4-KByte extent size uses 82 MBytes of in-memory metadata, the dirty parts of which are flushed to disk every 30 seconds. We evaluate *Clotho* using both microbenchmarks (Bonnie++ version 1.02 [4] and an in-house developed microbenchmark) and real-life setups with production-level filesystems. The Bonnie++ benchmark is a publicly available filesystem I/O benchmark [4]. For the real-life setup we run the SPEC SFS V3.0 suite on top of two well-known Linux filesystems, Ext2FS, and the high-performance journaled ReiserFS [20]. In our results we use the label *Disk* to denote experiments with the regular disk, without the *Clotho* driver on top.

4.1. Bonnie++

We use the Bonnie++ microbenchmark to quantify the basic overhead of *Clotho*. The filesystem we use in all Bonnie++ experiments is the Ext2FS with a 4-KByte extent size. The size of the file to be tested is 2 GBytes with block sizes ranging from 1 KByte to 64 KBytes. We measure accesses to the latest version of a volume with the following operations:

- *Block Write:* A large file is created using the `write()` system call.
- *Block Rewrite:* Each block of the file is read with `read()`, dirtied, and rewritten with `write()`, requiring an `lseek()`.
- *Block Read:* The file is read using a `read()` for every block.
- *Random Seek:* Processes running in parallel are performing `lseek()` to random locations in the file and `read()`ing the corresponding file blocks.

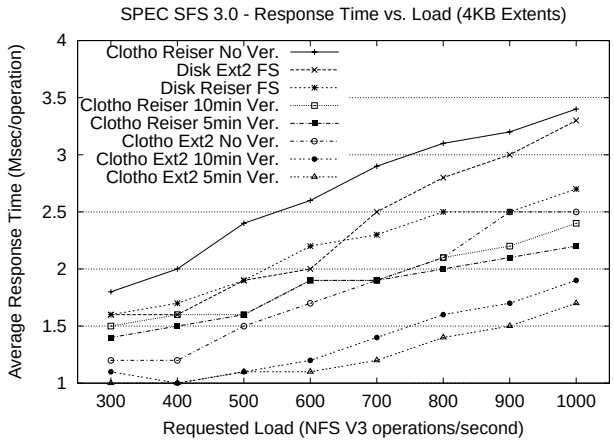


Figure 9: SPEC SFS response time using 4-KByte extents.

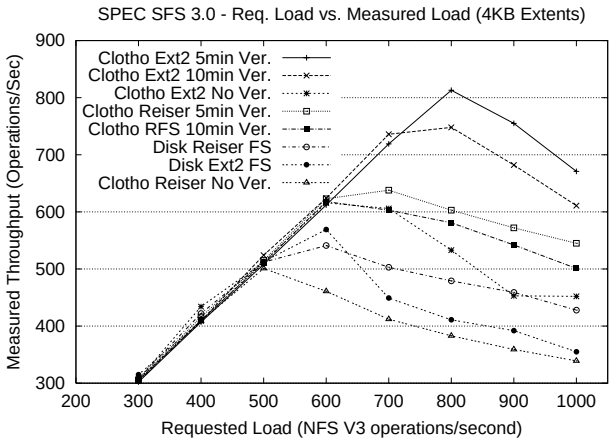


Figure 10: SPEC SFS throughput using 4-KByte extents.

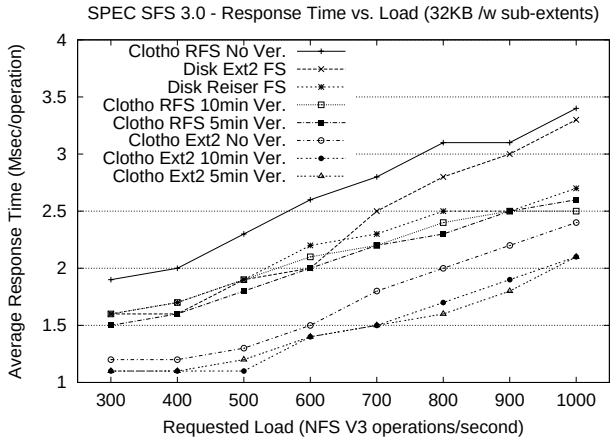


Figure 11: SPEC SFS response time using 32-Kbyte extents with subextents (RFS denotes ReiserFS).

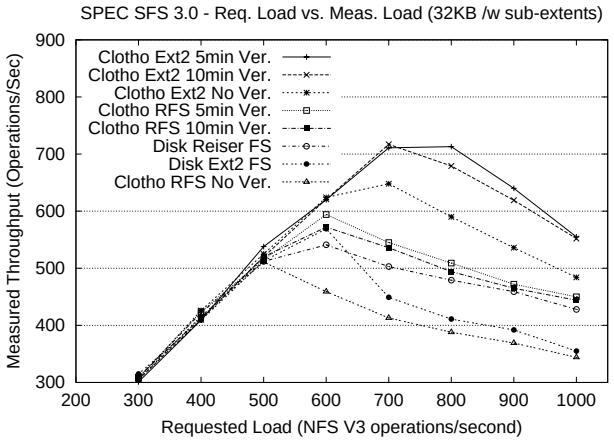


Figure 12: SPEC SFS throughput using 32-Kbyte extents with subextents (RFS denotes ReiserFS).

Figure 7 shows that the overhead in write throughput is minimal and the two curves are practically the same. In the read throughput case, *Clotho* performs slightly better than the regular disk. We believe this is due to the logging (sequential) disk allocation policy that *Clotho* uses. In the rewrite case, the overhead of *Clotho* becomes more significant. This is due to the random "seek and read" operation overhead, as shown in Figure 8. Since the seeks in this experiment are random, *Clotho*'s logging allocation has no effect and the overhead of translating I/O requests and flushing filesystem metadata to disk dominates. Even in this case, however, the overhead observed is at most 7.5% of the regular disk.

4.2. SPEC SFS

We use the SPEC SFS 3.0 benchmark suite to measure NFS file server throughput and response time over *Clotho*. We use one NFS client and one NFS server. The two systems that serve as client and server are connected with a switched 100 MBit/s Ethernet network. We use the following settings: NFS version 3 protocol over UDP/IP, one NFS exported directory, `biod_max_read=2, biod_max_write=2,` and requested loads ranging from 300 to 1000 NFS V3 operations/s with a 100 increment step. Both warm-up and run time are 300 seconds for each run and the time for all the SPEC SFS runs in sequence is approximately 3 hours. As mentioned above, we report results for the Ext2FS and ReiserFS (with -notail option) filesystems [20]. A new filesystem is created before every experiment.

We conduct four experiments with SPEC SFS for each of the two filesystems: Using the plain disk, using *Clotho* over the disk without versioning, using *Clotho* and versioning every 5 minutes, and using *Clotho* with 10 minute versioning. Versioning is performed throughout the entire 3 hour run of SPEC SFS. Figures 9 and 10 show our throughput and latency results for 4-Kbyte extents, while Figures 11 and 12 show the results using 32-KByte extents with subextent addressing.

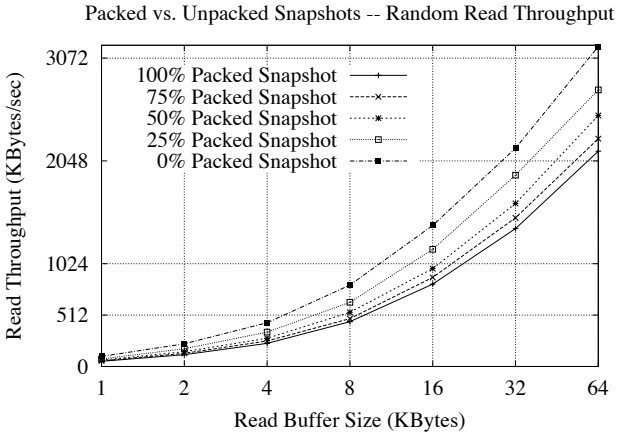


Figure 13: Random "compact-read" throughput.

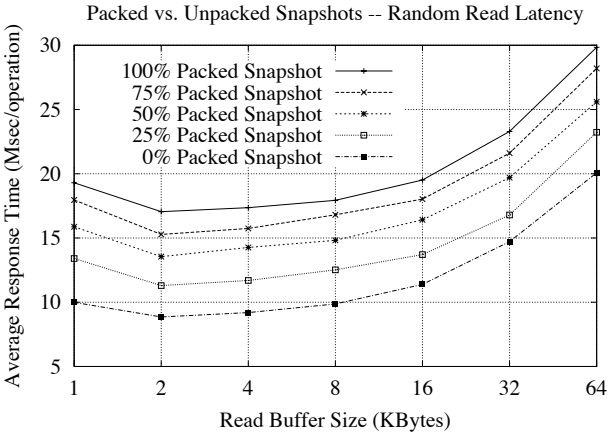


Figure 14: Random "compact-read" latency.

Our results show that *Clotho* outperforms the regular disk in all cases except ReiserFS without versioning. The higher performance is due to the logging (sequential) block allocation policy that *Clotho* uses. This explanation is reinforced by the performance in the cases where versions are created periodically. In this case, frequent versioning prevents disk seeks caused by overwriting of old data, which are now written to new locations on the disk in a sequential fashion. Furthermore, we observe that the more frequent the versioning, the higher the performance. The 32-KByte extent size experiments (Figures 11 and 12) show that even with much lower memory requirements, subextent mapping offers almost the same performance as the 4-KByte case. We attribute this small difference to the disk rotational latency, when skipping unused space to write subextents, while in the 4-KByte extent size, the extents are written "back-to-back" in a sequential manner.

Finally, comparing the two filesystems, Ext2 and ReiserFS, we find that the latter performs worse on top of *Clotho*. We attribute this behavior to the journaled metadata management of ReiserFS. While Ext2 updates metadata in place, ReiserFS appends metadata updates to a journal. This technique in combination with *Clotho*'s logging disk allocation appears to have a negative effect on performance in the SPEC SFS workload, compared to Ext2.

4.3. Compact version performance

Finally, we measure the read throughput of compacted versions to evaluate the space-time tradeoff of diff-based compaction. Since old versions are only accessible in read-only mode, we developed a two-phase microbenchmark that performs only `read` operations. In the first stage, our microbenchmark writes a number of large files and captures multiple versions of the data through *Clotho*. In writing the data the benchmark is also able to control the amount of similarity between two versions, and thus, the percentage of space required by compacted versions. In the second stage, our benchmark `mount`s a compacted version and performs 2000 random read operations on the files of the compacted version. Before every run, the benchmark flushes the system's buffer cache.

Figures 13 and 14 present latency and throughput results for different percentages of compaction. For 100% compaction, the compacted version takes up minimal space on the disk, whereas in the 0% case compaction is not space-effective at all. The difference in performance is mainly due to the higher number of disk accesses per read operation required for compacted versions. Each such read operation requires two disk reads to reconstruct the requested block. One read to fetch the block of the previous version and one to fetch the diffs. In particular, with 100% compaction, each and every read results in two disk accesses and thus, performance is about half of the 0% compaction case.

5. Related Work

A number of projects have highlighted the importance and issues in storage management [13, 16, 22, 36]. Our goal in this work is to define innovative functionality that can be used in future storage protocols and APIs to reduce management overheads.

Block-level versioning was recently discussed and used in WAFL [14], a file system designed for Network Appliance's NFS appliance. WAFL works in the block-level of the filesystem and can create up to 20 snapshots of a volume and keep them available online through NFS. However, since WAFL is a filesystem and works in an NFS appliance, this approach depends on the filesystem. In our work we demonstrate that *Clotho* is filesystem agnostic

by presenting experimental data with two production-level filesystems. Moreover, WAFL can manage a limited number of versions (up to 20), whereas *Clotho* can manage a practically unlimited number. The authors in [14] mention that WAFL's performance cannot be compared to other general purpose file systems, since it runs on a specialized NFS appliance and much of its performance comes from its NFS-specific tuning. The authors in [15] use WAFL to compare the performance of filesystem- and block-level-based snapshots (within WAFL). They advocate the use of block-level backup, due to cost and performance reasons. However, they do not provide any evidence on the performance overhead of block-level versioned disks compared to regular, non-versioned block devices. In our work we thoroughly evaluate this with both microbenchmarks as well as standard workloads. SnapMirror [23] is an extension of WAFL, which introduces management of remote replicas in WAFL's snapshots to optimize data transfer and ensure consistency.

Venti [27] is a block-level network storage service, intended as a repository for backup data. Venti follows a write-once storage model and uses content based addressing by means of hash functions to identify blocks with identical content. Instead, *Clotho* uses differential compression concepts. Furthermore, Venti does not supper versioning features. *Clotho* and Venti are designed to perform complementary tasks, the former to version data and the latter as a repository to store safely the archived data blocks over the network. Distributed block-level versioning support was included in Petal [7]. Although the concepts are similar to *Clotho*, Petal also targets networks of workstations as opposed to active storage devices.

Since backup and archival of data is an important problem, there are many products available that try to address the related issues. However, specific information about these systems and their performance with commodity hardware, filesystems, or well-known benchmarks are scarce. LiveBackup [29] captures changes at the file level on client machines and sends modifications to a back-end server that archives previous file versions. EMC's SnapView [8] runs on the CLARiiON storage servers at the block level and uses a "copy-on-first-write" algorithm. However, it can capture only up to 8 snapshots and its copy algorithm does not use logging block allocation to speed up writes. Instead, it copies the old block data to hidden storage space on every first write, overwriting another block. Veritas's FlashSnap [35] software works inside the Veritas File System, and thus, unlike *Clotho*, is not filesystem agnostic. Furthermore it supports only up to 32 snapshots of volumes. Sun's Instant Image [31] works also at the block-level in the Sun StorEdge storage servers. Its operation appears similar to *Clotho*. However, it is used through drivers and programs in the Sun's StorEdge architecture, which runs only through the Solaris architecture and is also filesystem aware.

Each of the above systems, especially the commercial ones, uses proprietary customized hardware and system software, which makes comparisons with commodity hardware and general purpose operating systems difficult. Moreover, these systems are intended as standalone services within centralized storage appliances, whereas *Clotho* is designed as a transparent autonomous block-lever layer for active storage devices and appropriate for pushing functionality closer to the physical disk. In this direction, *Clotho* categorizes the challenges of implementing block-level versioning and addresses the related problems.

The authors in [6] examine the possibility of introducing an additional layer in the I/O device stack to provide certain functionality at lower system layers, which also affect the functionality that is provided by the filesystem. Other efforts in this direction, mostly include work in logical volume management and storage virtualization that try to create a higher level abstraction on-top of simple block devices. The authors in [32] present a survey of such systems for Linux. Such systems usually provide the abstraction of a block-level volume that can be partitioned, aggregated, expanded, or shrunk on demand. Other such efforts [18] add RAID capabilities to arbitrary block devices. Our work is complementary to these efforts and proposes adding versioning capabilities to the block-device level.

Other previous work in versioning data has mostly been performed either at the filesystem layer or at higher layers. The authors in [26] propose versioning of data at the file level, discussing how the filesystem can transparently maintain file versions as well as how these can be cleaned up. The authors in [19] try to achieve similar functionality by providing mount points to previous versions of directories and files. They propose a solution that does not require kernel-level modification but relies on a set of user processes to capture user requests to files and to communicate with a back-end storage server that archives previous file versions. Other, similar efforts [21, 24, 25, 28, 30] approach the problem at the filesystem level as well and either provide the ability for checkpointing of data or explicitly manage time as an additional file attribute.

Self-securing storage [30] and its filesystem, CVFS [28] target secure storage systems and operate at the filesystem level. Some of the versioning concepts in self-securing storage and CVFS are similar to *Clotho*, but there are numerous differences as well. The most significant one is that self-securing storage policies are not intended for data archiving and thus, retain versions of data for a short period of time called *detection window*. No versions are guaranteed to exist outside this window of time and no version management control is provided for specifying

higher-level policies. CVFS introduces certain interesting concepts for reducing metadata space, which however, are also geared towards security and are not intended for archival purposes. Since certain concepts in [28, 30] are similar to *Clotho*, we believe that a block-level self-secure storage system could be based on *Clotho*, separating the orthogonal versioning and security functionalities in different subsystems.

6. Limitations and Future work

The main limitation of *Clotho* is that it cannot be layered below abstractions that aggregate multiple block devices in a single volume and cannot be used with shared block devices transparently. If *Clotho* is layered below a volume abstraction that performs aggregation and on top of the block devices that are being aggregated in a single volume, policies for creating versions need to perform synchronized versioning accross devices to ensure data consistency. However, this may not be possible in a transparent manner to higher system layers. The main issue here is that it is not clear what are the semantics of versioning parts of a "coherent", larger volume. Furthermore, when multiple clients have access to a shared block device, as is usually the case with distributed block devices [7, 33], *Clotho* cannot be layered on top of the shared volume in each client, since internal metadata will become inconsistent accross *Clotho* instances. Solutions to these problems are interesting topics for future work.

Another limitation of *Clotho* is that it imposes a change in the block layout from the input to the output layer. *Clotho* acts as a filter between two block devices, transferring blocks of data from one layer to the next. Although this does not introduce any new issues with wasting space due to fragmentation (e.g. for files if a filesystem is used with *Clotho*), it alters significantly the data layout. Thus, it may affect I/O performance, if free blocks are scattered over the disk or if higher layers rely on a specific block mapping, e.g. block 0 being the first block on the disk, block 1 the second, etc. However, this is an issue not only with *Clotho*, but with any layer in the I/O hierarchy that performs block remapping, such as RAIDs and some volume managers. Moreover, as I/O subsystems become more complex and provide more functionality, general solutions to this problem may become necessary. Since this is beyond the scope of this work, we do not discuss this any further here.

7. Conclusions

Storage management is an important problem in building future storage systems. Online storage versioning can assist reduce these costs directly, by addressing data archival and retrieval costs and indirectly, by providing novel storage functionality. In this work we propose pushing versioning functionality closer to the disk and implementing it at the block-device level. This approach takes advantage of technology trends in building active self-managed storage systems to address issues related to backup and version management.

We present a detailed design of our system, *Clotho*, that provides versioning at the block-level. *Clotho* imposes small memory and disk space overhead for version data and metadata management by using large extents, sub-extent addressing and diff-based compaction. It imposes minimal performance overhead in the I/O path by eliminating the need for copy-on-write even when the extent size is larger than the disk-block size and by employing logging (sequential) disk allocation. It provides mechanisms for dealing with data consistency and allows for flexible policies for both manual and automatic version management.

We implement our system in the Linux operating system and evaluate its impact on I/O path performance with both microbenchmarks as well as the SPEC SFS standard benchmark on top of two production-level file systems, ExtFS and ReiserFS. We find that the common path overhead is minimal for read and write I/O operations when versions are not compacted. For compact versions, the user has to pay the penalty of double disk accesses for each I/O operation that accesses a compact block.

Overall, we believe that our approach is promising in offloading significant management overhead and complexity from higher system layers to the disk itself and is a concrete step towards building self-managed storage devices.

8. Acknowledgments

We thankfully acknowledge the support of Natural Sciences and Engineering Research Council of Canada, Canada Foundation for Innovation, Ontario Innovation Trust, the Nortel Institute of Technology, Communications and Information Technology Ontario, Nortel Networks and the General Secretariat for Research and Technology, Greece.

References

[1] M. Ajtai, R. Burns, R. Fagin, D. Long, and L. Stockmeyer. Compactly Encoding Unstructured Inputs with Differential Compression. *Journal of the ACM*, 39(3), 2002.

[2] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running Circles Around Storage Administration. In *Proceedings of the FAST '02 Conference on File and Storage Technologies (FAST-02)*, pages 175–188, Berkeley, CA, Jan. 28–30 2002. USENIX Association.

[3] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, 1994.

[4] R. Coker. Bonnie++. http://www.coker.com.au/bonnie++.

[5] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making Backup Cheap and Easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI-02)*, Berkeley, CA, Dec. 9–11 2002. The USENIX Association.

[6] W. de Jonge, M. F. Kaashoek, and W. C. Hsieh. The Logical Disk: A New Approach to Improving File Systems. In *Proc. of 14th SOSP*, pages 15–28, 1993.

[7] Edward K. Lee and Chandramohan A. Thekkath. Petal: Distributed Virtual Disks. In *Proceedings of ASPLOS VII*, Oct. 1996.

[8] EMC. Snapview data sheet. http://www.emc.com/pdf/ products/ clariion/SnapView2_DS.pdf.

[9] S. C. Esener, M. H. Kryder, W. D. Doyle, M. Keshner, M. Mansuripur, and D. A. Thompson. WTEC Panel Report on The Future of Data Storage Technologies. International Technology Research Institute. World Technology (WTEC) Division, June 1999.

[10] GartnerGroup. Total Cost of Storage Ownership – A User-oriented Approach, Sept. 2000.

[11] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. In *Proc. of the 8th ASPLOS*, Oct. 1998.

[12] G. A. Gibson and J. Wilkes. Self-managing network-attached storage. *ACM Computing Surveys*, 28(4es):209–209, Dec. 1996.

[13] J. Gray. What Next? A Few Remaining Problems in Information Technology (Turing Lecture). In *ACM Federated Computer Research Conferences (FCRC)*, May 1999.

[14] D. Hitz, J. Lau, and M. Malcolm. File System Design for an NFS File Server Appliance. In *Proceedings of the Winter 1994 USENIX Conference*, pages 235–246, 1994.

[15] N. C. Hutchinson, S. Manley, M. Federwisch, G. Harris, D. Hitz, S. Kleiman, and S. O'Malley. Logical vs. Physical File System Backup. In *Proc. of the 3rd USENIX Symposium on Operating Systems Design and Impl. (OSDI99)*, Feb. 1999.

[16] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*, November 2000.

[17] M. Lesk. How Much Information Is There In the World? http://www.lesk.com/ mlesk/ ksg97/ ksg.html, 1997.

[18] M. Icaza and I. Molnar and G. Oxman. The Linux RAID-1,-4,-5 Code. In *LinuxExpo*, Apr. 1997.

[19] J. Moran, B. Lyon, and L. S. Incorporated. The Restore-o-Mounter: The File Motel Revisited. In *Proc. of USENIX '93 Summer Technical Conference*, June 1993.

[20] Namesys. Reiserfs. http://www.namesys.com.

[21] M. A. Olson. The Design and Implementation of the Inversion File System. In *Proc. of USENIX '93 Winter Technical Conference*, Jan. 1993.

[22] D. Patterson. The UC Berkeley ISTORE Project: bringing availability, maintainability, and evolutionary growth to storage-based clusters. http://roc.cs.berkeley.edu, January 2000.

[23] R. H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara. SnapMirror: File-System-Based Asynchronous Mirroring for Disaster Recovery. In *Proceedings of FAST '02*. USENIX, Jan. 28–30 2002.

[24] R. Pike, D. Presotto, K. Thompson, and H. Trickey. Plan 9 From Bell Labs. In *Proc. of the Summer UKUUG Conference*, 1990.

[25] W. D. Roome. 3DFS: A Time-Oriented File Server. In *Proceedings of USENIX '92 Winter Technical Conference*, Jan. 1992.

[26] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, and J. Ofir. Deciding When to Forget in the Elephant File System. In *Proceedings of 17th SOSP*, Dec. 1999.

[27] Sean Quinlan and Sean Dorward. Venti: A New Approach to Archival Data Storage. In *Proceedings of FAST '02*, pages 89–102. USENIX, Jan. 28–30 2002.

[28] C. A. Soules, G. R. Goodson, J. D. Strunk, and G. R. Ganger. Metadata Efficiency in Versioning File Systems. In *Proceedings of the FAST '03 Conference on File and Storage Technologies (FAST-03)*, Berkeley, CA, Apr. 2003. The USENIX Association.

[29] Storactive. Delivering real-time data protection & easy disaster recovery for windows workstations. http://www.storactive.com/files/Storactive_Whitepaper.doc, Jan. 2002.

[30] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-Securing Storage: Protecting Data in Compromised Systems. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI-00)*, pages 165–180, Berkeley, CA, Oct. 23–25 2000.

[31] Sun Microsystems. Instant image white paper. http://www.sun.com/storage/white-papers/ii_soft_arch.pdf.

[32] D. Teigland and H. Mauelshagen. Volume managers in linux. In *Proceedings of USENIX 2001 Technical Conference*, June 2001.

[33] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A Scalable Distributed File System. In *Proceedings of the 16th SOSP*, volume 31 of *Operating Systems Review*, pages 224–237, New York, Oct. 5–8 1997. ACM Press.

[34] A. C. Veitch, E. Riedel, S. J. Towers, and J. Wilkes. Towards Global Storage Management and Data Placement. In *Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 184–184. IEEE Computer Society Press, May 20–23 2001.

[35] Veritas. Flashsnap. http://eval.veritas.com/ downloads/ pro/ fsnap_guide_wp.pdf.

[36] J. Wilkes. Traveling to Rome: QoS specifications for automated storage system management. In *Proc. of the Int. Workshop on QoS (IWQoS'2001). Karlsruhe, Germany*, June 2001.

[37] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23:337–343, May 1977.

US NATIONAL OCEANOGRAPHIC DATA CENTER ARCHIVAL MANAGEMENT PRACTICES AND THE OPEN ARCHIVAL INFORMATION SYSTEM REFERENCE MODEL

Donald W. Collins
US Department of Commerce/NOAA/NESDIS
National Oceanographic Data Center
1315 East West Highway, SSMC3 Fourth Floor
Silver Spring, MD 20910-3282
Donald.Collins@noaa.gov
+1-301-713-3272
+1-301-713-3302

Abstract
This paper describes relationships between the Open Archival Information System Reference Model (OAIS RM) and the archival practices of the NOAA National Oceanographic Data Center (NODC). The OAIS RM defines a thorough approach to defining the processes, entities, and framework for maintaining digital information in an electronic archival environment without defining how to implement the framework. The NODC Archival Management System (AMS) is an example of an implementation of a persistent digital archive. Major OAIS RM components, such as the Submission Information Package, Archival Information Package, Dissemination Information Package, and Archival Storage are clearly comparable between the OAIS RM and the NODC AMS. The main participants (Producer, Consumer, Management, and OAIS) are represented in the NODC AMS, as are many primary functions (Ingest Process, Archive Process, Dissemination Process). Some important OAIS RM components, such as a consistent Submission Agreement and a deeper level of Preservation Description Information may be missing for some of the information archived in the NODC AMS. It is instructive to document the commonalities between the NODC system and the OAIS RM as the NOAA National Data Centers expand archival services for a broad and growing range of digital environmental data.

1 Introduction
Imagine that you are on a large ship on a calm sea with no land in sight and 3 miles of water beneath your feet. You are a marine biologist studying the habitat of the giant squid, so you want to know the range of water temperatures and salinity values, the available nutrients eaten by local microscopic organisms, what those organisms are and what visible-size sea life might be eating them. These pieces of information can be collected by a host of instruments you deploy over the side of the ship and lower through the water. The ship stops periodically to collect these measurements, which are electronically recorded into a series of data files that you will use later in the lab to characterize the conditions in which the giant squid lives.

If you received funding for your research from the US Federal government, you are likely to be contractually obligated to send a copy of the data you collected to the National Oceanographic Data Center (NODC), along with enough descriptive metadata to make the data meaningful to others. The NODC is one of several data centers operated by the

U.S. Department of Commerce National Oceanic and Atmospheric Administration (NOAA).

The NODC is believed to archive the largest collection of *in situ* measurements of oceanographic parameters in the world, with approximately 300 gigabytes of data, metadata, and model output stored in its digital archives (D. Knoll, NODC, pers. comm., 2003). The NODC has recently undertaken a substantial effort to update and improve the archival management of these electronic environmental records and to improve access to the original data records in its collection. This paper examines how these new record management strategies and procedures relate to the Open Archival Information System Reference Model (OAIS RM) [1], ISO 14721:2002, which defines the major elements and functions of an electronic records archive. To explore the relationship between the OAIS RM and the new NODC Archives Management System (AMS), the first part of this paper provides an overview of the main features of the OAIS RM. The second part of the paper describes in some detail the NODC AMS in terms of the conceptual elements of the OAIS Reference Model. The last part of the paper discusses OAIS RM elements that are not yet implemented by the NODC AMS.

2 OAIS Reference Model

The OAIS Reference Model was initially developed by the Consultative Committee for Space Data Systems to identify the essential elements needed by an electronic records archive to manage records over long time scales [2]. The OAIS RM acknowledges technological changes that inevitably create substantial challenges for electronic records archives [3]. As described by Lavoie [4], an OAIS means "any organization or system charged with the task of preserving information over the Long Term and making it accessible to a specified class of users (known as the Designated Community)." In keeping with the practice established in the OAIS RM documentation, entities and procedures named in the model and discussed below will have initial capital letters (e.g., Producer). By definition in the OAIS RM (p. 1-11), data are archived for the Long Term, which is defined as "[a] period of time long enough for there to be concern about the impacts of changing technologies, including support for new media and data formats, and of a changing user community, on the information being held in a repository. This period extends into the indefinite future."

The main participants in any OAIS are defined as Producer, Consumer, Management, and the OAIS archive functions. In the OAIS RM, Producer represents the people or systems that create and/or provide information to be preserved in the archive. Consumer represents the people or systems that use the OAIS to find and retrieve preserved information. A Designated Community may be a special subset of the domain of all Consumers. Management is the person or group that sets the policy for the management of the OAIS, among other activities: it is not responsible for the day-to-day operations of the OAIS. The OAIS (archive) is the system that provides for the long term storage, migration, and dissemination of the information that is archived. Using this model and the example in the Introduction to this paper, the following roles could be identified:

- Producer: Ocean scientists (e.g., squid specialist) making measurements of physical, chemical, and biological conditions;

- Consumer (Designated community): other ocean scientists, resource managers, general public;
- Management: US National Oceanographic Data Center Director and Deputy Director;
- OAIS (archive): US National Oceanographic Data Center Archive Management System.

One of the basic concepts of the Reference Model is that information is a combination of data and its Representation Information. Regardless of whether the data part of the information is physical (e.g., a giant squid) or digital (e.g., digital photographs of the giant squid), Representation Information allows the Consumer to fully interpret the meaning of the information. For digital objects, Representation Information typically includes some mapping of bits into recognizable data types (e.g., characters, integers). It also associates these mappings into higher-level groupings of data types, which are called Structure Information. To fully understand how to interpret the Structure Information, it is important to include Semantic Information, which defines the language of the Structure Information [5].

Another main component of the OAIS RM is the Information Object, which is comprised of the following components:

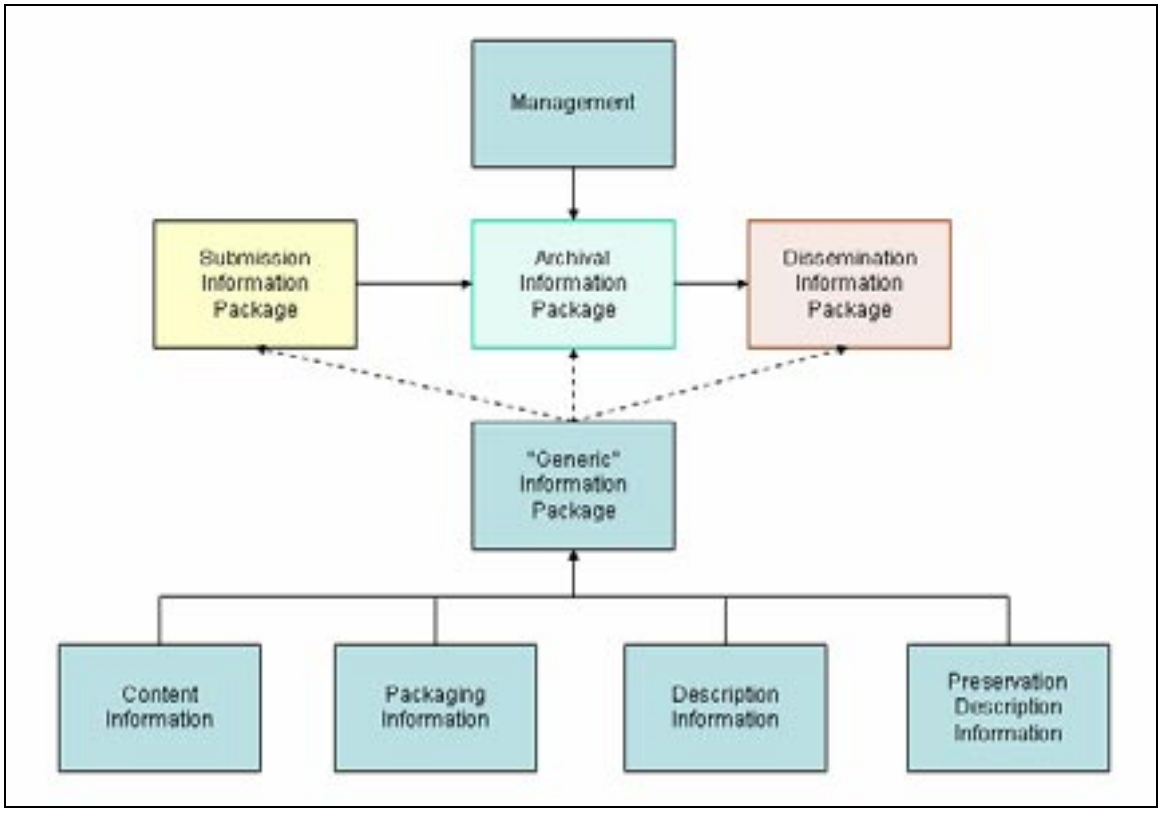
- Content Information object, which is equivalent to the contents of a letter, email, observed data values, etc.;
- Packaging Information object, which is information that wraps the different information objects into a cohesive bundle;
- Descriptive Information object, which provides descriptive content and contextual information about the Content Information object;
- Preservation Description Information object, which includes significant information about the electronic form and structure of the bits that can be translated into the Content Information object. It also includes information about provenance and authenticity validation characteristics of the Content Information object.

The OAIS RM distinguishes between Information Objects based on the role that the Information Object performs in the information management process. Three specific types of Information Object are designated: the Submission Information Package (SIP), the Archival Information Package (AIP), and the Dissemination Information Package (DIP). Each Information Package contains the four information objects defined above. Figure 1 depicts the relationship between the components of an Information Package and between SIP, AIP, DIP, and Management (after Sawyer [6]).

A Producer creates electronic records: the contents are the property of the Producer and may be in any format that is deemed useful by the Producer. Once the Producer decides to transfer the data to the OAIS, a Submission Agreement is negotiated between Producer and OAIS to define the terms of the information transfer. The OAIS may help the Producer in developing the SIP by providing information, tools, or other assistance in preparing the contents of the SIP, especially the Preservation Description Information

component. The SIP contains Content Information, but must also contain sufficient metadata to ensure that the Content Information can by maintained properly by the OAIS and be used by future Consumers [7]. The SIP is then transferred to the OAIS in one or more Data Submission Sessions.

Fig. 1. OAIS Reference Model Information Package components (After Sawyer, 2002.)



Upon receipt of the SIP, the OAIS creates an Archival Information Package (AIP) using the archives' ingest procedures. The OAIS defines the appropriate ingest procedures for creating the AIP according to the archives policies and guidelines. The OAIS may modify the form and content of the SIP: "An OAIS is not always required to retain the information submitted to it in precisely the same format as in the SIP. Indeed, preserving the original information exactly as submitted may not be desirable [8]". The intention here is to ensure the preservation of digital information, not to modify or tamper with the digital content. Once the SIP is transformed into the corresponding AIP and Package Descriptions (i.e., the information needed to make an AIP accessible from appropriate Access Aids) during the ingest process, the AIP is stored in an Archival Storage entity. Ingest processes, Package Descriptions, and Archival Storage hardware and software may vary significantly from one OAIS to another.

The OAIS determines how to make the AIPs in its collection available to its Designated Community. To perform this function, OAIS defines a Dissemination Information

Package (DIP). The DIP is described in the Package Description, found using Access Aids provided by the OAIS. Access Aids are the tools provided to discover and obtain AIPs from the OAIS. Finding Aids, Collection Descriptions, Ordering Aids and other data discovery tools are types of Access Aids. When a user discovers the existence of archived materials through available Access Aids, the selected AIPs are assembled into a Dissemination Information Package (DIP) and transferred to the Consumer via a Data Dissemination Session. The structure and mechanism for delivering a DIP to Consumers depends on the way the archival organization creates its DIPs [9].

This introduction to the main components of an OAIS only scratches the surface of an extremely detailed and well-defined set of terms, objects, concepts, and procedures that an electronic archives needs to address to ensure the preservation of data and information for an indeterminate "Long Term". The next section of this paper describes the archival practices established by the US NODC and relates many of the functions, processes and information objects of the new NODC Archives Management System to the components of the OAIS Reference Model.

3 NODC Archives Management SystemA Case Study

The US National Oceanographic Data Center (NODC) was created as an inter-Departmental support organization administered by the US Navy in 1960 and transferred to the Environmental Science Services Administration (ESSA) in the mid-1960s. When the National Oceanic and Atmospheric Administration (NOAA) was created in 1970, the NODC became one of the three environmental data centers administered by the NOAA. The NODC receives oceanographic data from a diverse community of international oceanographic organizations, government organizations (federal, state, and local), and public and private universities and research institutions from around the world.

The primary commonality among these organizations and the data they provide to the NODC archives is that the information is somehow related to the world's oceans, seas, and coastal areas. Data formats and structures, languages used, and the types of data collected can be extremely variable. Most oceanographic data collected in the past decade or two are provided to NODC in digital data files. As technology changes, new equipment for obtaining, storing and organizing measurements are invented. Data formats and structures change to accommodate new measurable values and techniques. The only constant throughout these technological changes is the need for accurate metadata about the instruments used (e.g., calibration and methodologies), data format structure and other documentation.

By 1967, the NODC recognized the need to improve tracking of data that were sent to the NODC. A data set identification system was developed in which groups of data were assigned an NODC Accession Number to identify the information as a unit. An accession is loosely defined as 'a logical grouping of related data,' which is usually interpreted as a group of data that are received together. Typical examples of the types of data in an accession include: *in situ* water column measurements (e.g., water temperature and salinity, nutrient concentrations such as dissolved nitrate and silicate, current speed and

direction), biological observations (e.g., abundance and taxonomic identification of plankton and fish species), or satellite observations of sea surface characteristics.

As of December 31, 2003, there were 20,419 individual accessions in the NODC archives. New accessions are presently acquired at a rate of about 30 per month. During a recent month, more than 62 Gigabytes of data were downloaded by 1088 individual host computers connected to online services that access NODC information, holdings and products created from those holdings (NODC Information Systems and Management Division, unpublished report).

The NODC recently developed the NODC Archives Management System (AMS) to bring the electronic and analog data, metadata, and administrative information files for oceanographic data collections into a more robust and flexible environment. The Accession Tracking Data Base (ATDB), the Archive File Management System (AFMS), the Ocean Archive System and the NODC Metadata Repository (NMR) are the primary components of the Archives Management System. The NODC Metadata Repository is a commercially-available database (using a proprietary structure and database developed by Blue Angel Technologies, Inc. and Oracle Corp.) that is optimized for managing data set descriptions in the Federal Geographic Data Committee (FGDC) Content Standard for Digital Geospatial Metadata (CSDGM) structure. While this component will soon provide descriptive metadata to assist with search and retrieval processes, the underlying software and database of this component of the AMS are beyond the scope of this paper. The emphasis in the next few sections is to describe the AMS components designed, developed and deployed within NODC on generic computer hardware using open source software for the operating system, database design and management, and preliminary data entry, search, and retrieval requirements.

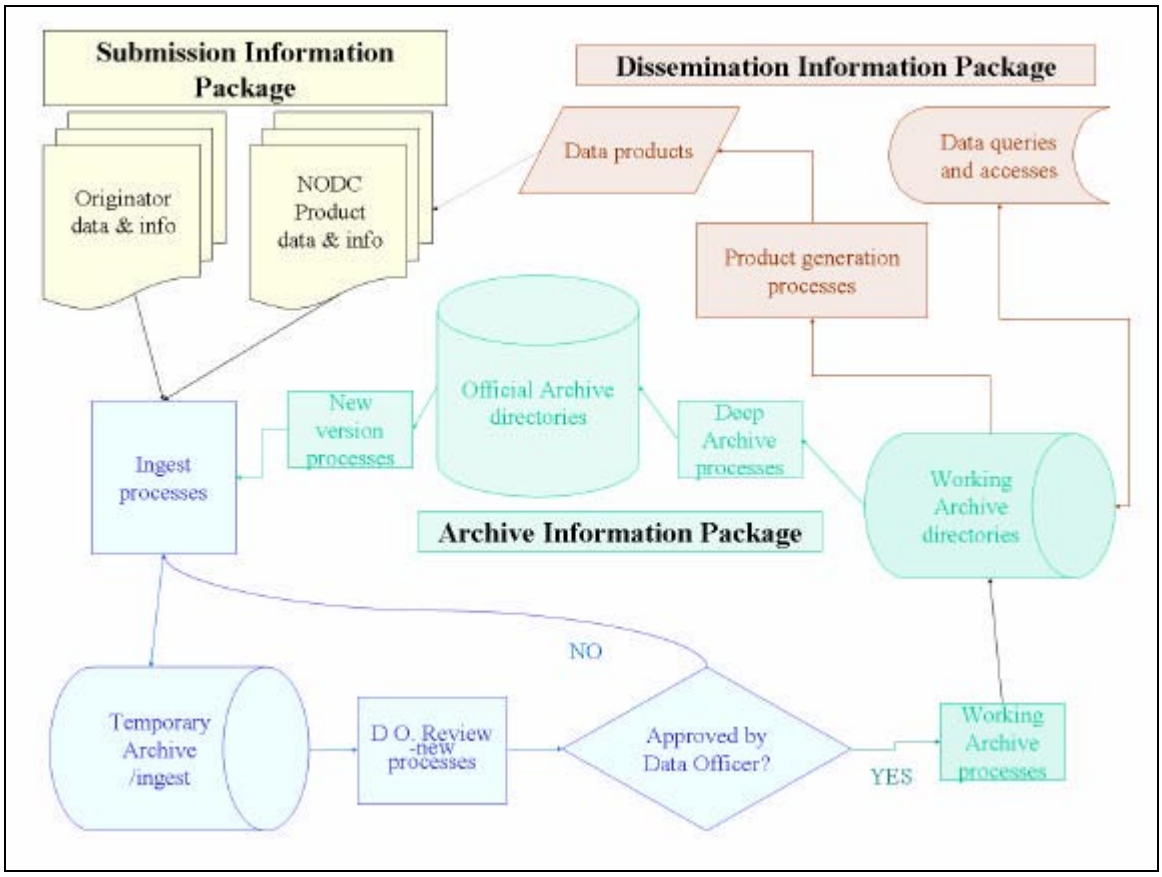
The NODC AMS identifies a series of information elements, procedures, and standardized practices that facilitate ingesting, describing, accessioning, storing, migrating, and accessing archived digital records and information. These functions are discussed below and are related to relevant OAIS RM concepts or constructs. Figure 2 depicts the generic flow of an Information Package through the Archives Management System.

3.1 Ingest Procedures

The NODC frequently works with data Producers to prepare their information for archiving, i.e., to create a more meaningful Submission Information Package. At the NODC, the SIP is equivalent to a single accession as it is received from the Producer (internally, NODC currently refers to "originators data" rather than "a SIP"). During this interaction, the NODC encourages the data Producer to provide the most generic representation of their data possible and to provide as much descriptive documentation as possible. The reason for this is to obviate the need for NODC to undertake after-the-fact translation tasks to represent data that are in software and/or platform dependent data structures into less dependent structures. In many instances, these translations can be undertaken with little or no loss of meaning for the data. The NODC prefers to have the

data Producer make these translations whenever possible, rather than making such translations during the ingest process.

Fig. 2. NODC AMS flow diagram for an Information Package.



As shown in Figure 2, a Producer collects data and prepares it in some fashion for shipment to the NODC. Of course, the method of "shipment" and the medium used in this Data Submission Session has changed substantially over time from mailing analog records, computer punch cards, or magnetic tapes to FTP transfers, web site downloads, or sending CD-ROMs or DVDs. Ingest procedures now in place for each Data Submission Session are:

- obtain and review the data/information files and/or other information objects in the SIP from the Producer;
- create a new record in the NODC ATDB (described below), which establishes the 'canonical structure' for the accessioned SIP in a preparation area;
- copy/move digital data files to the appropriate directory of the Archives File Management System (described below);
- request that the ATDB record be closed;
- review of 'closed' accession record and SIP by 'Data Officer';
- transfer approved SIP to the archival storage area (creation of the AIP).

Once a SIP is ingested and accessioned and an AIP is created, the data in it may be processed into an NODC product (e.g., the NODC World Ocean Database, which contains more than 7 million temperature, salinity, and other parameter profiles) or delivered as a direct copy (DIP) using the NODC Ocean Archive System, an online discovery and retrieval service (OAIS Access Aid) described in greater detail below.

3.2 Accession Tracking Data Base

The Accession Tracking Data Base (ATDB) is a relational database that supports initial SIP ingest procedures and administrative metadata management. The ATDB uses postgreSQL database management software on a generic server platform running the Linux operating system. Generically-designed browser-based user interfaces were developed using perl/CGI scripts to facilitate data entry and limited search/retrieval capabilities. Linux, postgreSQL, and perl are all freely available open source software designed to operate on a number of hardware platforms. One of the guiding principles for developing the ATDB and AMS was to determine if open source software tools were robust enough to support a mission critical information management system, rather than relying on commercial packages that require continuous license upkeep, maintenance and other expenses. The initial results suggest that the open source tools are more than adequate for the purposes of creating a workable file management system and a database to manage thousands of digital files. The ATDB represents several components of the Administration function in the OAIS RM [10].

A 'Brief Access Record' (BAR) is created for each new SIP using the main table in the ATDB. The BAR allows a new entry to be made in the ATDB based on a relatively few administrative and descriptive metadata elements. Administrative metadata elements keep track of an accession in the NODC File Management System and other pieces of internal information of importance to NODC but that may not necessarily be useful for Consumers. Administrative metadata, combined with descriptive metadata elements (discussed later) are roughly equivalent to the Package Description in the OAIS RM. Administrative metadata elements in the ATDB are:

- accession number (unique identifier for an Archival Information Package, automatically generated by the database),
- date received (date the SIP was received at the NODC),
- keyer, editor (name of the NODC employee creating the ATDB entry and making the most recent change to the ATDB entry, selected from a controlled vocabulary)
- keydate, editdate (system-assigned date-time stamp of ATDB entry creation and latest ATDB entry change),
- status (information indicator that denotes the 'state' of the Archival Information Package, i.e., 'new', 'archived', 'revision'),
- version (information indicator that denotes the most recent version of the AIP),
- availability date (date after which this AIP can be made accessible to the public; default value is the same as the 'date received'),
- requested action (information indicator to request action from the Data Officer function, i.e., 'close' ('assess this SIP and metadata for inclusion in the archives') or 'open' ('check the AIP out of the archive for revision, creating a new version of the AIP'),

- NODC contact (name of the NODC employee who is most familiar with the contents of this SIP or AIP. May be different than the 'keyer' or 'editor').

The ATDB also requires the inclusion of a minimal amount of descriptive metadata about the SIP. Available descriptive information is passed from the ATDB via a program to the NODC Metadata Repository, creating a partially-complete FGDC CSDGM-structured description of this AIP. Descriptive metadata elements in the ATDB include a title, start and end dates, latitude and longitude bounding coordinates, and controlled-vocabulary descriptors for institution names, sea areas, parameter and observation types, instrument types, project names, and platform (ship) names.

About 13 staff members are designated as keyers which means that they can create the initial BAR in the ATDB, although the majority of new BARs are created (at present) by 3 or 4 people. Four senior, non-management staff members are designated Data Officers. The Data Officers are responsible for reviewing the work of the keyers, approving new accessions for inclusion in the archives, and developing related archival management policy recommendations for approval by NODC Management.

3.3 NODC Archives File Management System

Creating a new record in the ATDB causes a perl script to create a directory structure on a storage disk (represented in Figure 2 as "Temporary Archive" and referred to internally as the "ingest area" or "/ingest") in the canonical directory structure (Table 1). While still in the ingest area, it is possible to modify or add to the contents of a SIP. Once the contents of the SIP are finalized, a program is run that copies files from the ingest area to the "archive area" or "/archive". This Archival Storage is presently provided by a large capacity RAID device with limited write privileges, i.e., only the archivist function (in most cases, via a program) can write files to this system. The canonical structure of an NODC AIP is represented in Table 1.

Table 1. NODC AMS File Management System canonical structure. Elements listed in *italics* are only part of the final archival version of the file structure. A "/" indicates the listed element is a directory.

7-digit unique number/	NODC Accession number (e.g., 0000001)
0000001.01-version.md5	*File containing checksum values for all files in this AIP.*
1-version/	*Directory identifying the most recent version of this AIP, beginning with 1-version.*
NODC_ReadMe.txt	Text file that describes this directory structure.
about/	Directory for storing information files created by NODC about this AIP.
journal.txt	NODC-created file describing actions taken by NODC staff regarding this AIP.
other files…	Optional other files created and/or maintained that are not part of the data in this AIP.
data/	Directory for storing the original data and translations of the original data in the AIP.

0-data/	Directory for storing an exact copy of the original files in this AIP.
directories or files	Copy of the original files obtained from the Producer.
1-data/	Directory for storing translated versions of the original files in this AIP.
directories or files	Translated versions of the original files in this AIP in a directory structure that may resemble the original directory structure.

The main sections of the canonical AIP structure are /about and /data. Any additional information about the SIP, such as emails between NODC and Producer or other NODC-created information about the SIP, are placed in the /about subdirectory. The journal.txt file, which is initially created along with the directory structure, is used to document any steps taken by NODC personnel while performing ingest processes on the SIP. At present, NODC guidelines require that an exact copy of the original files in a SIP are copied to the /0-data directory, regardless of their original file structure. The /1-data directory is where NODC-created translations of the original SIP contents may be placed. The intention of this directory is to provide a place for non-proprietary representations of proprietary original data from the SIP. A typical example of the contents of /1-data might be the comma-separated ASCII representation of the text and values from a Microsoft Excel spreadsheet file. In an effort to minimize the probable lack of access to this file due to program changes or loss of vendor support for the program in the future, NODC attempts to translate files from proprietary structures to generic structures.

3.4 Archiving Procedures

The NODC continues to refine its archiving procedures for digital accessions. Archiving processes are undertaken by the employees designated as Data Officer. As noted above, Data Officer tasks (see Figure 2) include reviewing the work of the keyers and determining if an accession is ready to be moved from the /ingest area to the /archive area. There are currently only four criteria that the Data Officer uses to make this determination:

- there is a fully-populated ATDB entry,
- the journal.txt file has been updated,
- the files in the /ingest directory are in the canonical form (Table 1),
- a "reasonable attempt" has been made to translate data from proprietary formats to generic formats and translations are placed in /1-data.

When a keyer determines that the ingest-AIP needs no further action, the ATDB element ‘requested action’ is set to ‘Close’. This indicates to the Data Officer that the SIP is ready for review. If the four criteria listed above are met, the Data Officer approves the transfer of the SIP from the ingest area to the Working Archive Directories (Archival Storage), creating the 'working archive' copy of the AIP (Figure 2). The program that transfers files from /ingest to /archive calculates a checksum value for each file, which can be used for future validation of the contents of the AIP, and also runs additional virus-detection software to minimize the chance of archiving a virus with the AIP.

As noted in Figure 2, there are also "Deep Archive processes" that include the creation and validation of off-site copies of the AIP (one is maintained in Asheville, NC at the National Climatic Data Center and one will soon be maintained at the NODC National Coastal Data Development Center in Stennis Space Center, MS). These 'deep archive' copies are intended for use in disaster recovery situations or when the local 'working archive' copy is rendered temporarily unavailable due to equipment malfunction or other reasons. This backup process represents the Replication function described in the OAIS RM [11].

Figure 2 also shows a 'New Version processes' step. These processes, which require approval from the Data Officer, are used when a new version of an existing AIP is required. New versions are occasionally requested by a Producer, usually when an error is found in a previously submitted data set. In cases where a new version is requested, an exact copy of the existing AIP is 'checked out' of the archive area and placed back in the ingest area. Modifications are then made to the data files and/or metadata files by the editor who requested the new version. When all modifications have been made, the same approval process is followed by the Data Officer, with special attention paid to the documentation of what modifications were made, why they were made, and who made them. The entire AIP is then 'checked-in' to the Working Archive Directories as "/2-version" (or the next available version number). In this fashion, NODC maintains a copy of each iteration of a specific data set. Circumstances for determining if an update SIP should be a new version of an existing AIP or should be a new AIP are decided on a case-by-case basis by the Data Officer and Management. The current philosophy adopted by NODC management is that it is better to keep everything, including obsolete versions, than to be unable to provide back to the originator an exact copy of everything that was given to the Center.

3.5 Metadata Management

Descriptive metadata for new AIPs, in the form of FGDC CSDGM-compliant database records, are presently created and/or maintained in two places: the ATDB Brief Access Record and the NODC Metadata Repository (NMR). The BAR is discussed in detail above. The NMR is beyond the scope of this paper, but will be used to manage additional descriptive metadata for each AIP in the NODC collection. There is some overlap between the information in each of these databases. NODC maintains several authority tables (e.g., people, ship names, institutions, place names) in the ATDB to facilitate the creation of consistent descriptions of an AIP. Controlled vocabulary entries will only be updated through the ATDB. Details of how to propagate any updates efficiently into the NMR are still in development. Ultimately, NODC plans to maintain a comprehensive accession tracking and descriptive metadata database from which any number of formatted descriptive information (e.g., FGDC, ISO 19115, Dublin Core) could be exported. As mentioned above, descriptive metadata is part of the OAIS concept of Package Description.

The OAIS RM requires a substantial level of semantic and syntactic metadata to be maintained in the Preservation Description Information (PDI) object of the AIP. At present, this level of metadata and the existence of a PDI object in an AIP depend on the

inclusion of such information in the SIP from the Producer. NODC is investigating how to create or refine the contents of this critical information element.

3.6 Data Dissemination

The NODC distributes two general types of Dissemination Information Package (DIP). One type of DIP is an exact copy of an AIP, with no additional processing beyond the possible translations described above. The other type of DIP distributed by the NODC is a Derived AIP, typically data from multiple AIPs processed by a product developer (usually within NODC, but occasionally by an external organization) to create a value-added data set, referred to as an NODC Standard Product. In most cases, all of the data in an NODC Standard Product have been reformatted to a single data format and possibly had some type of quality control checks performed, such as marking values that erroneously appear in a land area or a measured value that is outside the range of possible values for a parameter. The obvious advantage of these products is that similar types of data that were originally in a variety of data structures may now be easily inter-compared or otherwise statistically manipulated.

The OAIS RM categorizes Access Aids as a Finding Aid or an Ordering Aid. A Finding Aid allows a consumer to search for and discover DIPs that are of interest to the consumer. Ordering Aids are applications that help the Consumer to obtain DIPs of interest and include information costs and other handling circumstances that may be needed to transfer a copy of the DIP to the consumer. Standard products from the NODC (primarily Derived AIPs that are copied to CD-ROM or DVD and mass-replicated) can be discovered through a number of Finding Aid-like applications (e.g., NOAAServer, an FGDC Clearinghouse application), ordered from the NOAA National Data Centers Online Store [12], or discovered and downloaded using the NODC Ocean Archive System (OAS) [13].

The OAS is the current interface available directly from the NODC for searching, discovering, and retrieving copies of original data from the AMS. The OAS interface presents several of the ATDB descriptive metadata elements in a tabular form and allows a Consumer to select one or more descriptive elements to create a database query. Once the query has been processed, a list of accession numbers and descriptors for AIPs that match the query are presented, in addition to links to the full ATDB record and to the File Management System working archive directory. The Consumer can assess the relevance of the contents of the AIP and save all or part of the AIP to a local directory.

A small reference team is available to assist a Consumer in deciding whether there is a Standard Product that is more suitable for their needs or to assist with finding appropriate AIPs. In general, the majority of requests for data and information are satisfied by a Standard Product. However, as AIPs become readily available online DIPs through the OAS or other Access Aids, the demand for additional assistance for Consumers in deciding about appropriate data is expected to increase. In general, the Consumers for data from the NODC are: ocean scientists of all types from all over the world and all levels of government (the Designated Community), non-scientist business persons (often

lawyers and insurance representatives) and the general public (usually for K-12 educational purposes or recreational purposes).

4 What is missing?

The NODC Archives Management System described above is still in development, although the ATDB and File Management System parts of the AMS have been used operationally since about April 2002. The Ocean Archives System Finding Aid was released for public use in December 2003. Additional descriptive metadata elements were added to the ATDB in mid-2003 to accommodate metadata from a legacy database. But as noted above, the links between the ATDB and the NODC Metadata Repository (NMR) are not yet fully developed, although the same information used to load the ATDB with legacy information was used to populate the NMR for each AIP. Many issues related to maintaining the referential integrity of the information that resides in both the ATDB and the NMR are still being examined.

The discussion above outlines how the main elements of the NODC Archives Management System map to many of the major components of the OAIS Reference Model. However, some important OAIS RM components are missing from the NODC AMS. In particular, two areas that need additional work are the development of a Submission Agreement and identifying Detailed Preservation Information for each AIP in the AMS.

In most cases, no officially-communicated Submission Agreement spells out the terms, conditions, and other responsibilities of the NODC to act as the Long Term custodian of an Information Package. The NODC negotiates and maintains data and information exchange agreements with a number of US and foreign organizations as a routine part of its archives efforts, but many data sets are submitted to the NODC with little more than an email from the Producer or designated intermediary and varying levels of descriptive metadata. It is unclear if these sometimes informal cover letters are sufficient to serve as an OAIS Submission Agreement. The NODC participates in the development of the NOAA Comprehensive Large Array-data Stewardship System (CLASS), which is developing a Submission Agreement on behalf of the NOAA National Data Centers [14]. The draft version of the CLASS Submission Agreement is not yet available for public review, but currently is modeled using the FGDC Content Standard for Digital Geospatial Metadata (CSDGM) as a framework for defining a variety of custodial descriptive metadata. It is not yet clear if a separate formal Submission Agreement will be developed to authorize transferring data and information from the Producer to the NODC in addition to the current draft Submission Agreement.

Perhaps the most difficult missing element in the NODC AMS is the availability of detailed Preservation Description Information. The OAIS RM defines this as "[t]he information which is necessary for adequate preservation of the Content Information and which can be categorized as Provenance, Reference, Fixity, and Context Information [15]". Some of this missing information will be provided for future accessions if something like the draft CLASS Data Submission form accompanies each Submission Information Package, but what about the historic data that already reside in the NODC

collections? Approximately 2700 of the 20,000+ AIPs archived at NODC have some type of information about their provenance and context in a Data Documentation Form (DDF), which was a standard form used to document data submitted to the NODC from the early 1970s until the mid-1990s. The quantity and quality of the information in the DDFs varies significantly and is dependent on the effort made by the Producer to document the data; nearly all existing DDFs are analog documents stored either on-site at NODC or in an off-site storage facility. In OAIS terms, most historic AIPs at NODC have little or no level of Fixity Information (which "…authenticates that the Content Information has not been altered in an undocumented manner [16]"). Likewise, there are very few instances where Representation Information (i.e., "information that maps a Data Object into more meaningful concepts... [such as] the ASCII definition that describes how a sequence of bits is mapped into a symbol [17]") is present at all. The NODC is beginning to consider how to address these difficult information deficiencies as part of the planning to maintain its collections of data for the Long Term.

5 Conclusions

The Open Archival Information Systems Reference Model describes a very thorough approach to defining the processes, entities, and framework for maintaining digital information in a electronic archival environment without defining how to implement the framework. The NODC Archival Management System provides an example of how to implement a persistent digital archive for oceanographic data. Many of the processes and components correlate well with elements of the OAIS Reference Model. Major components, such as the Submission Information Package, Archival Information Package, Dissemination Information Package, and Archival Storage are clearly comparable between the OAIS RM and the NODC AMS. The main participants (Producer, Consumer, Management, and OAIS) are also all present in the NODC AMS, as are many of the primary functions (Ingest Process, Archive Process, Dissemination Process). On the other hand, the NODC AMS is frequently lacking some important OAIS RM components, such as a consistent Submission Agreement and a deeper level of Preservation Description Information.

This paper establishes a relationship between the OAIS Reference Model and the archival management practices of the NOAA National Data Centers. It is important to document the commonalities in the NODC system and the OAIS RM as NOAA and the NOAA National Data Centers continue to develop and upgrade archival services for a broad and growing range of digital environmental data. While there are many commonalities between the NODC AMS and the OAIS RM, the NODC (and by extension, the NOAA National Data Centers) needs to be aware of the types of information that are not presently available or actively acquired for data that are sent to be archived. It is imperative that these environmental data records not become the "write once, read never" records bemoaned by Barkstrom [18] because they will provide the baseline scientific data for future environmental investigations and future generations.

6 References

[1] Consultative Committee for Space Data Systems, 2002, Reference Model for an Open Archival Information System OAIS. Available online at

http://wwwclassic.ccsds.org/documents/pdf/CCSDS-650.0-B-1.pdf (last accessed April 2003).
[2] Lavoie, Brian, 2000, Meeting the challenges of digital preservation: The OAIS reference model. OCLC Newsletter, No. 243 (January/February 2000), p. 26-30. Available online at http://www.oclc.org/research/publications/newsletter/repubs/lavoie243/ (last accessed April 2003).
[3] Consultative Committee for Space Data Systems, 2002, p. 2-1.
[4] Lavoie, Brian, 2000, p. 27.
[5] Consultative Committee for Space Data Systems, 2002, p. 4-21.
[6] Sawyer, Donald, 2002, ISO "Reference Model for an Open Archival Information System (OAIS)": Tutorial Presentation. Presentation to University of Maryland College of Information Studies, October 2002, 31p.
[7] Consultative Committee for Space Data Systems, 2002, p. 4-49.
[8] Consultative Committee for Space Data Systems, 2002, p. 4-49.
[9] Consultative Committee for Space Data Systems, 2002, p. 4-52.
[10] CCSDS, 2002, p. 1-7 and p. 4-10.
[11] CCSDS, 2002, p. 5-5.
[12] NOAAServer is available online at http://www.esdim.noaa.gov/noaaserver-bin/NOAAServer (last accessed January 2004). The NOAA National Data Centers Online Store is available online at http://www.nndc.noaa.gov/dev/prototype/nndcserver/nndchome.html (last accessed January 2004).
[13] NODC Ocean Archive System can be accessed at http://www.nodc.noaa.gov/search/prod/ (last accessed January 2004).
[14] Habermann, Ted, (in prep.), Comprehensive Large Array-data Stewardship System (CLASS) Data Product Submission Agreements, 20p.
[15] CCSDS, 2002, p. 1-12.
[16] CCSDS, 2002, p. 1-10.
[17] CCSDS, 2002, p. 1-13.
[18] Barkstrom, Bruce R., 1998, Digital archive issues from the perspective of an Earth science data producer. NASA Technical Report, NASA Langley Research Center Atmospheric Sciences Division, Available online from http://techreports.larc.nasa.gov/ltrs/papers/NASA-98-dadw-brb/ (last accessed April 2003).

Acknowledgement
The author would like to thank Lauren Brown and the students in the "Seminar in Archives, Records, and Information Management" course at the University of Maryland College of Information Studies for their constructive comments. Also, many thanks to Kurt Schnebele, Tony Picciolo, Steve Rutz, Mary Lou Cumberpatch, Anna Fiolek, Bob Gelfeld and Donna Collins for their support, constructive observations and suggestions.

Storage Resource Sharing with CASTOR

Olof Bärring, Ben Couturier, Jean-Damien Durand, Emil Knezo, Sebastien Ponce
CERN
CH-1211 Geneva 23, Switzerland
Olof.Barring@cern.ch, Ben.Couturier@cern.ch, Jean-Damien.Durand@cern.ch,
Emil.Knezo@cern.ch, Sebastien.Ponce@cern.ch
tel: +41-22-767-3967
fax: +41-22-767-7155

Vitaly Motyakov
Institute for High Energy Physics
RU-142281, Protvino, Moscow region, Russia
motyakov@mx.ihep.su
tel: +7-0967-747413
fax: +7-0967-744937

Abstract:
The Cern Advanced STORage (CASTOR) system is a hierarchical storage management system developed at CERN to meet the requirements for high energy physics applications. The existing disk cache management subsystem in CASTOR, the *stager*, was developed more than a decade ago and was intended for relatively moderate (large at the time) sized disk caches and request load. Due to internal limitations a single CASTOR stager instance will not be able to efficiently manage distributed disk cashes of several PetaBytes foreseen for the experiments at the Large Hadron Collider (LHC) which will be commissioned in 2007. The Mass Storage challenge comes not only from the sheer data volume and rates but also from the expected request load in terms of number of file opens per second. This paper presents the architecture design for a new CASTOR stager now being developed to address the LHC requirements and overcome the limitations with the current CASTOR stager.

Efficient management of PetaByte disk caches made up of clusters of 100s of commodity file servers (e.g. linux PCs) resembles in many aspects the CPU cluster management, for which sophisticated batch scheduling systems have been available since more than a decade. Rather than reinventing scheduling and resource sharing algorithms and apply them to disk storage resources, the new CASTOR stager design aims to leverage some of the resource management concepts from existing CPU batch scheduling systems. This has led to a pluggable framework design, where the scheduling task itself has been externalized allowing the reuse of commercial or open source schedulers.

The development of the new CASTOR stager also incorporates new strategies for data migration and recall between disk and tape media where the resource allocation takes place just-in-time for the data transfer. This allows for choosing the best disk and network resources based on current load.

1. Introduction

The Cern Advanced STORage (CASTOR) system[1] is a scalable high throughput hierarchical storage system (HSM) developed at CERN. The system was first deployed for full production use in 2001 and is now managing about 13 million files for a total data volume of more than 1.5 PetaByte. The aggregate traffic between disk cache and tape archive usually exceeds 100 MB/s (~75% tape read) and there are of the order of 50,000 – 100,000 tape mounts per week. CASTOR is a modular and fault-tolerant system designed for scalable distributed deployments on potentially unreliable commodity hardware. Like other HSM systems (see for instance [2], [3], [4], [5]) the client front-end for file access consists of a distributed disk cache with file servers managed by the CASTOR stager component, and an associated global logical name-space contained in an Oracle database (MySQL is also supported). The backend tape archive consists of a volume library (Oracle or MySQL database), tape drive queuing, tape mover and physical volume repository. The next section lists some of the salient features of today's CASTOR system and the installation at CERN. Thereafter are listed some requirements that the CASTOR system has to meet well before the experiments at the Large Hadron Collider (LHC) start their data taking in 2007.

The remaining sections (4 - 6) describe the new CASTOR stager (disk cache management component), which is being developed [6] to cope with the data handling requirements for LHC. It is in particular the expected request load (file opens) that requires some special handling. The design target is to be able to sustain *peak* rates of the order of 500 - 1000 file open requests per second for a PetaByte disk pool. The new developments have been inspired by the problems arising with management of massive installations of commodity storage hardware. It is today possible to build very large distributed disk cache systems using low cost Linux fileservers with EIDE disks. CERN has built up farms with several 100s of disk servers with of the order of 1TB disk space each. The farming of disk servers raises new problems for the disk cache management: request scheduling, resource sharing and partitioning, quality of service management, automated configuration and monitoring, and fault tolerance to unreliable hardware. Some of those problems have been addressed and solved by traditional batch systems for scheduling of CPU resources. With the new CASTOR stager developments described in this paper, the CASTOR team leverages the work already done for CPU scheduling and applies it for the disk cache resource management. This is achieved through pluggable framework design where the request scheduling is delegated to an external component. Initially LSF from Platform Inc [7] and Maui from Supercluster [8] will be supported.

The new system is still under development and only exists as an advanced prototype. The final system is planned for 2Q04.

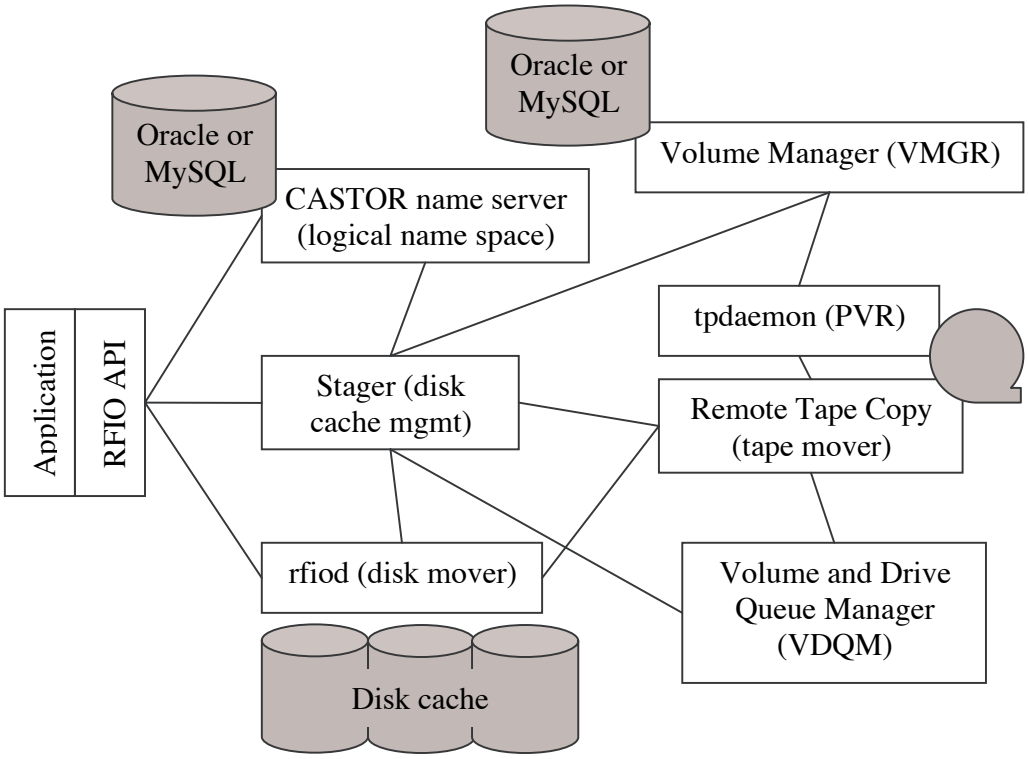


Figure 1: CASTOR components and their interactions

2. The CASTOR architecture and capabilities

The CASTOR software architecture is schematically depicted in Figure 1. The client application should normally use the Remote File IO (RFIO) library to interface to CASTOR. RFIO provides a POSIX compliant file IO and metadata interface, e.g. *rfio_open(), rfio_read(), rfio_write(), rfio_lseek(), rfio_stat()*, etc. For bulk operations on groups of files, a low-level stager API is provided, which allows for passing arrays of files to be processed.

The Name server provides the CASTOR namespace, which appears as a normal UNIX filesystem directory hierarchy. The name space is rooted with "/castor" followed by a directory that specifies the domain, e.g. "/castor/cern.ch" or "/castor/cnaf.infn.it". The next level in the name space hierarchy identifies the hostname (or alias) for a node with a running instance of the CASTOR name server. The convention is *nodename = "cns" + "directory name"*, e.g. "/castor/cern.ch/user" points to the CASTOR name server instance running on a node *cnsuser.cern.ch*. The naming of all directories below the third level hierarchy is entirely up to the user/administrator managing the sub-trees.

The file attributes and metadata stored in the CASTOR name space include all normal POSIX "stat" attributes. The CASTOR name server assigns a unique 64 bit file identifier to all name space elements (files and directories). File class metadata associates tape migration and recall policies to all files. These policies include:

- Number of tape drives to be used for the migration

- Number of copies required on different media. CASTOR supports multiple copies for precious data

In order to avoid waste of tape media, CASTOR supports segmentation of large files over several tapes. The tape metadata stored in the CASTOR name space are currently:

- The tape VID
- Tape position: file sequence number and blockid (if position by blockid is supported by the tape device)
- The size of the file segment
- The data compression on tape

There is also a 'side' attribute for (future) DVD or CD-RW support. The tape metadata will soon be extended to include the segment checksum. CASTOR does not manage the content of the files but a special "user metadata" field can be attached to the files in the CASTOR name space. The user metadata is a character string that the client can use for associating application specific metadata to the files.

The RFIO client library interfaces to three CASTOR components: the name server providing the CASTOR namespace described in previous paragraph; the CASTOR stager, which manages the disk cache for space allocations, garbage collection and recall/migration of tape files; the RFIO server (rfiod), which is the CASTOR disk mover and implements a POSIX IO compliant interface for the user application file access.

All tape access is managed by the CASTOR stager. The client does not normally know about the tape location of the files being accessed. The stager therefore interfaces with:

- The CASTOR Volume Manager (VMGR) to know the status of tapes and select a tape for migration if the client created a new file or updated an existing one
- The CASTOR Volume and Drive Queue Manager (VDQM), which provides a FIFO queue for accessing the tape drives. Requests for already mounted volumes are given priority in order to reduce the number of physical tape mounts
- The CASTOR tape mover, Remote Tape COPY (RTCOPY), which is a multithreaded application with large memory buffers, for performing the copy between tape and disk
- The RFIO server (rfiod) for managing the disk pools

The CASTOR software is an evolution of an older system, SHIFT, which was CERN's disk and tape management system for almost a decade until it was replaced by CASTOR in 2001. SHIFT was awarded '21st Century Achievement Award by Computerworld in 2001' [9].

The CASTOR software has been compiled and tested on a large variety of hardware: Linux, Solaris, AIX, HP-UX, Digital UNIX, IRIX and Windows (NT and W2K). The CASTOR tape software supports DLT/SDLT, LTO, IBM 3590, STK 9840, STK9940A/B tape drives and ADIC Scalar, IBM 3494, IBM 3584, Odetics, Sony DMS24, STK Powderhorn tape libraries as well as all generic SCSI driver compatible robotics.

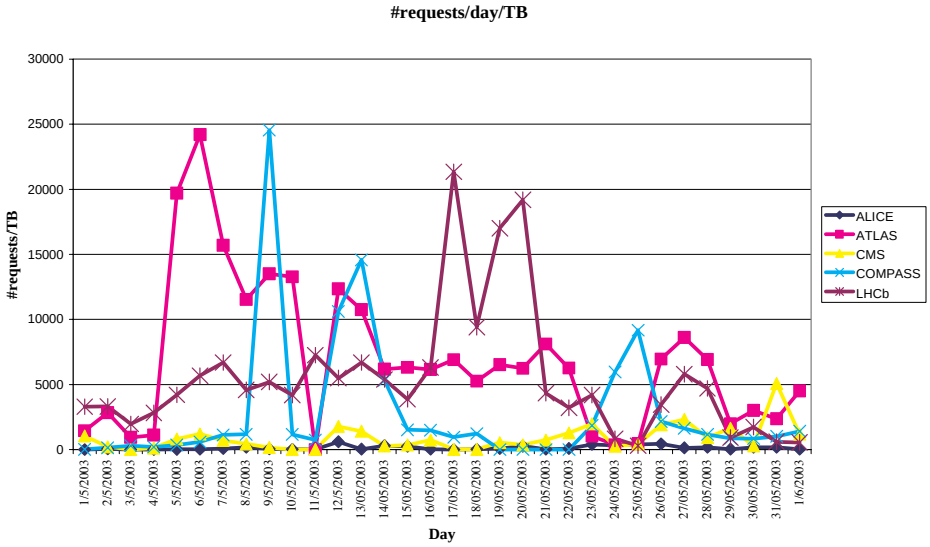


Figure 2: The number of requests per day normalized to TB of disk pool space. The plot shows one month of activity for 5 different stagers. ALICE, ATLAS, CMS and LHCb are the four LHC experiments whereas COMPASS is a running heavy ion experiment.

The CASTOR RFIO client API provides remote file access through both a POSIX I/O compliant interface and a data streaming interface. The former is usually used by random access applications whereas the latter is used by the tape mover and disk-to-disk copy. At the server side the data streaming mode is implemented with multiple threads overlaying disk and network I/O. When using the RFIO streaming mode the data transfer performance is normally only limited by hardware (see [10] for performance measurements). Parallel stream transfers are not supported since this is not required by High Energy Physics applications. An exception is the CASTOR GridFTP interface, which does support parallel stream transfers in accordance with the GridFTP v1.0 protocol specification.

3. Requirements for the LHC

CASTOR is designed to cope with the data volume and rates expected from the LHC experiments. This is frequently tested in so called 'data challenges' that the CERN IT department runs together with the experiments. In late 2002, the ALICE experiment ran a data challenge with a sustained rate of 280 MB/s to tape during a week [11]. The CERN IT department has also shown that the CASTOR tape archive can sustain 1 GB/s during a day [12].

In addition to the high data volume and rates generated by the LHC experiments, it is expected that the physics analysis of the data collected by LHC experiments will generate a very high file access request load. The four LHC experiments are large scientific collaborations of thousands of physicists. The detailed requirements for the LHC physics analysis phase are unknown but from experience with previous experiments it can be expected that

- The data access is random
- Only a portion of the data within a file is accessed
- The set of files required by an analysis application is not necessarily known
- A subset of the files are hit by many clients concurrently (hotspots)

The files must therefore be disk resident and since a substantial part of the physics analysis will take place at CERN it will result in a high request load on the CASTOR stager. Since the detailed requirements are unknown it is difficult to give an exact estimate for the expected request load on CASTOR. In order to obtain a design target the activity of five running instances of the current CASTOR stager was observed during a month, see Figure 2. The rates are normalized to the size of the disk pools in TB. The figure shows a peak rate of about 25,000 requests/day (0.3 requests/second) per TB disk. The average was about 10 times lower: 3,400 requests/day per TB disk. The associated data rate to/from the disk cache depends on the file sizes and the amount of data accessed for each file. The data volume statistics from the period shown in Figure 2 is no longer available but looking at a more recent period it was found that for the ATLAS stager the average data transfer per request is of the order 20MB (400GB for a day with 20,000 requests) whereas for LHCb it was only 3MB (50GB for a day with 17,000 requests).

Assuming that the number of file access requests scales linearly with the size of the disk cache, the peak request rate for a PetaByte disk cache would be of the order 300 requests/second. The objective for the new CASTOR stager is therefore to be able to handle peak rates of 500 – 1000 requests per second. Under such load it is essential that the new system is capable of request throttling, which is not the case for the current CASTOR stager. It is also known that the current CASTOR stager catalogue will not scale to manage more than ~100,000 files in the disk cache. Those shortcomings have already today led to a deployment of many CASTOR stager instances, each with its dedicated moderate size (5-10TB) disk cache. The disadvantages are manifold but most important are:

- Each stager instance has its dedicated disk cache and this easily leads to suboptimal use of the resources where one instance runs at 100% and lacks resources while another instance may be idle.
- The configuration management becomes complex since each stager instance has its own configuration and the clients must know which stager to use.

4. New CASTOR stager architecture

The new CASTOR stager, which is currently being developed at CERN, is specifically designed to cope with the requirements listed in the previous section. This section briefly describes some of the salient features of the new architecture.

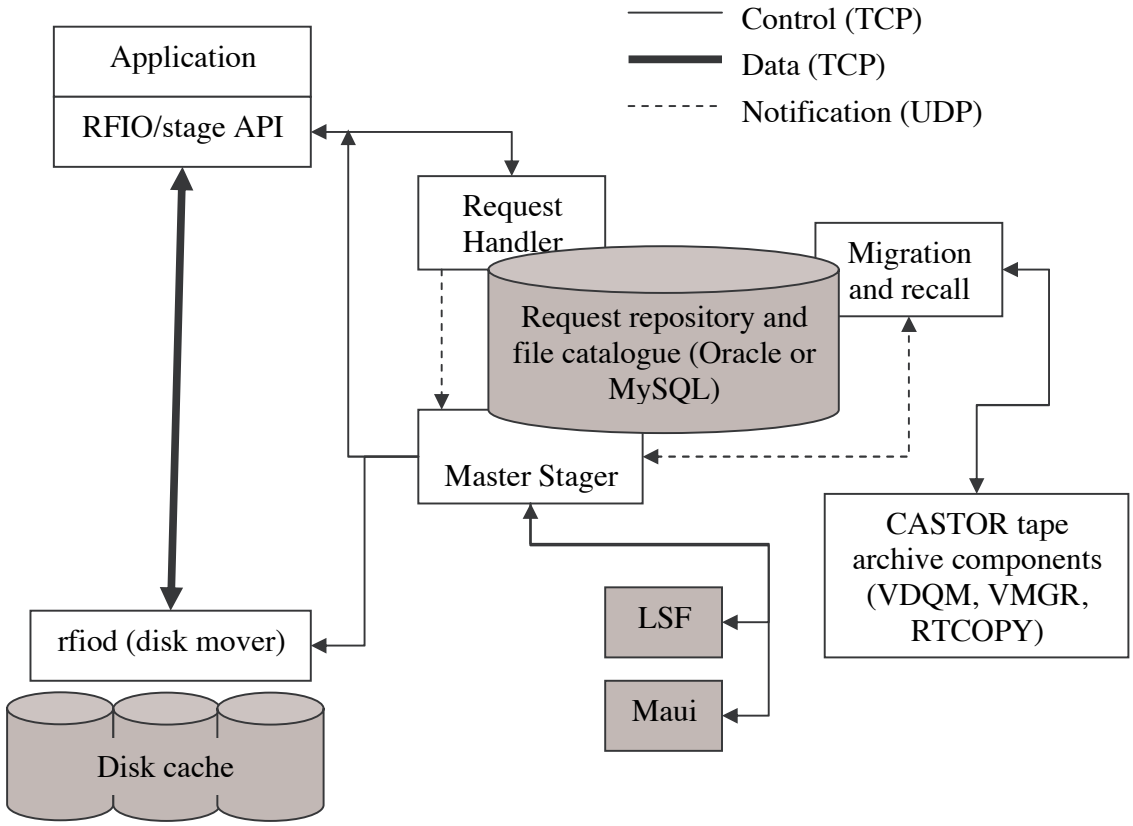


Figure 3: Overview of the new CASTOR stager architecture

4.1. Overview

Figure 3 shows a schematic view of the new architecture. The client application uses the RFIO API or the lower level stager API (for multi-file requests) to submit the file access request. Although it is not shown in the Figure 3, the CASTOR name server is called for all operations on the CASTOR name space as was shown in Figure 1. The request handler, master stager and migration/recall components communicate through a database containing all file access requests and the file residency on disk (catalogue). To avoid too frequent polling, unreliable notification is used between the modules when the database is being updated. The role of the request handler is to insulate the rest of the system from high requests bursts. The master stager is the central component that processes the file access requests and delegates the request scheduling decision to an external module. The first version of the new CASTOR stager will have resource management interfaces to LSF [7] and Maui [8]. The migration/recall components retrieve the tape related information and contacts the CASTOR tape archive (see Figure 1) to submit the tape mount requests.

Along with the new stager developments, it was decided to improve the security in CASTOR. However, for CERN physics applications, confidentiality is not a main requirement, and encrypting all the data during transfer would add significant load on the

CASTOR servers. Therefore, strong authentication of users in RFIO, stager and the CASTOR name server will be implemented using standard security protocols: Kerberos-5, Grid Security Infrastructure, and Kerberos-4 for compatibility with the current CERN infrastructure. Furthermore the architecture of the new stager and its interface with RFIO will allow only authorized clients (file access request has been scheduled to run) to access the files on the disk servers managed by the stager.

Below follows a more detailed description of the components and concepts used in the design of the new CASTOR stager architecture.

4.2. Database centric architecture

In the new CASTOR stager, a relational database is used to store the requests and their history, to ensure their persistency. The stager code is interfaced with Oracle and MySQL, but the structure allows plugging-in other Relational Database Servers if necessary. Furthermore, the database is also used for the communication between the different components in the stager (request handler, stager, migration/recall…)
This database centric architecture has many advantages over the architecture of the current stager.

From a developer's point of view:

- The data locking and concurrency when accessing the data are handled by the RDBMS: This makes the development/debugging of the code much easier.
- The database API is used to enter or read requests. The CASTOR application does not have to know whether the database is local or remote, the database API takes care of that (e.g. the Oracle API uses shared memory if the database is local or SQLNet otherwise).

From the CASTOR system's administrator point of view:

- The database being the central component in the architecture, the coupling between the different components is very low.
- This, as well as the persistence of the requests, and the proper use of status codes to identify their state, allows stopping and restarting the system, or specific components at any time. The administration is made easier.

For the CASTOR user, the overall scalability of the system is greatly improved as:

- RDBMS can cope with large amount of data. If the database tables are properly designed, the system should not have huge performance penalty when dealing with a large number of requests.
- The stager software components are stateless, which allows running several instances on different machines without problems. The real limit to the scalability is the database server itself.
- The database software is prepared for high loads, and there are some cluster solutions that can help coping with such problems.

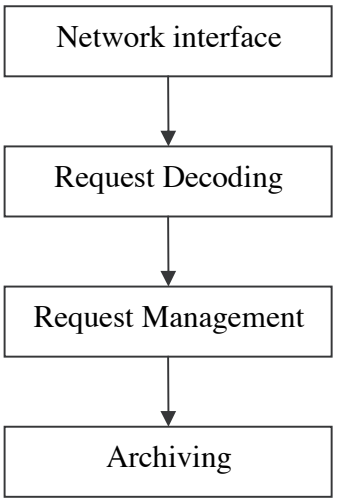


Figure 4: Request handler architecture

There are however some drawbacks to this architecture:

- The latency for each call is higher than with the current architecture, as the stager components have to access the database.
- The database server is a central point of failure and has to cope with a very high load. However, database technology is widely deployed and it is possible to get database clusters to mitigate that risk.

Furthermore, just using the database for communication between the components would force them to frequently poll the database, which creates unnecessary load, and possibly increase the latency of the calls to the stager. Therefore, in the current stager prototype, a notification mechanism using UDP messages has been introduced. This mechanism is very lightweight and does not create a big coupling between the different components as UDP is connectionless.

4.3. Request handler

The request handler handles all requests in CASTOR, from reception to archiving. The main purpose of the Request Handler is to shield the request scheduling from the possible request bursts. The request handler is also responsible for the decoding of the requests and storing them in the database. At all different stages in the request processing the requests are stored and updated in the database. The Request Handler architecture is sketched in Figure 4.

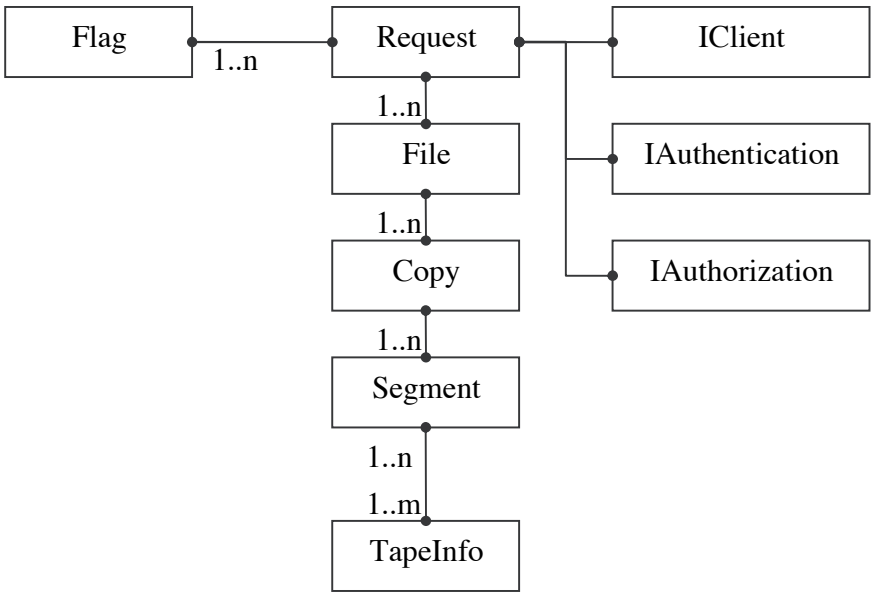


Figure 5: Request content

The **Network Interface** is responsible for handling new requests and queries coming from the clients. This is the component realizing the isolation against request bursts. It is designed to sustain high rates (up to 1000 requests per second). Therefore the only thing it does is to store the raw request as a binary object into a database for further processing by the request decoder. There are only two transactions on this table per request: one insert and one retrieve + delete. To guarantee the best insertion rate this database instance may be deployed separately from the rest of the request repository.

The **Request Decoding** takes place asynchronously. The decoded requests are stored into a database where they can be easily used and queried by the rest of the system, especially by the master stager scheduling component.

When a request is selected by the scheduling component for processing, it is moved to the **Request Management** components, which associate data to each request describing its status (flags), its clients (authenticated and authorized) and the tapes possibly required (see Figure 5).

At last, when the processing of a request is over, it is archived by the **Archiving** component.

The request handler manages two interfaces: the network interface that is used by external clients, and the interface to the Request Manager for internal communication with the other components of CASTOR:

- The network interface component allows clients to send requests to CASTOR. The API used is very much inspired by the SRM standard [13] so it will not be described in any detail here. Command line scripts are also provided.
- The internal interface has two parts:
    - The first part allows the stager component to handle requests and to update their metadata
    - The second part is used by the migration/recall component to get the list of tapes and segments of file to be migrated and to update the tape status.

4.4. Master stager and externalized scheduling plug-in interface

The master stager is central in the new architecture. It has two main functions:

- Manage all resident files and space allocations in the disk cache
- Schedule and control all client access to the disk cache

The master stager maintains a catalogue with up-to-date information of all resident files and space allocations in the disk cache. The catalogue maintains a mapping between CASTOR files (paths in the CASTOR name space) and their physical location in the disk cache. The mapping may be one-to-many since, for load-balancing purpose there may be several physical replicas of the same CASTOR file. An important difference to the current CASTOR stager is that the catalogue is implemented with a relational database. Oracle and MySQL are supported but other RDBMS systems can easily be interfaced. This assures that the system will scale to large disk caches with millions of files.
When the master stager gets notifications from the request handler that there are new requests to process, it uses the request handler internal interfaces to retrieve the requests from the database. The basic processing flows in the master stager are depicted in Figure 6:

- If the request is for reading or updating an existing CASTOR file, the master stager checks if the requested file is already cached somewhere in the disk cache. In case the file is already cached, the request is submitted to the external scheduler and queued for access to the file in the disk cache.
- If the request is for creating a new file, the entry is created in the CASTOR name space and the request is submitted to the external scheduler like in the previous case.
- If the request is for reading or updating an existing CASTOR file, which is not already cached, the master stager marks the file for recall from tape and notifies the recaller. Once the file has been recalled to disk the request is submitted to the external scheduler like before.

The scheduling of the access depends on the load (CPU, memory, I/O) on the file server and disk that holds the file. If the load is too high but another file server is ready to serve the request, an internal disk-to-disk copy may take place replicating the file to the selected disk server.

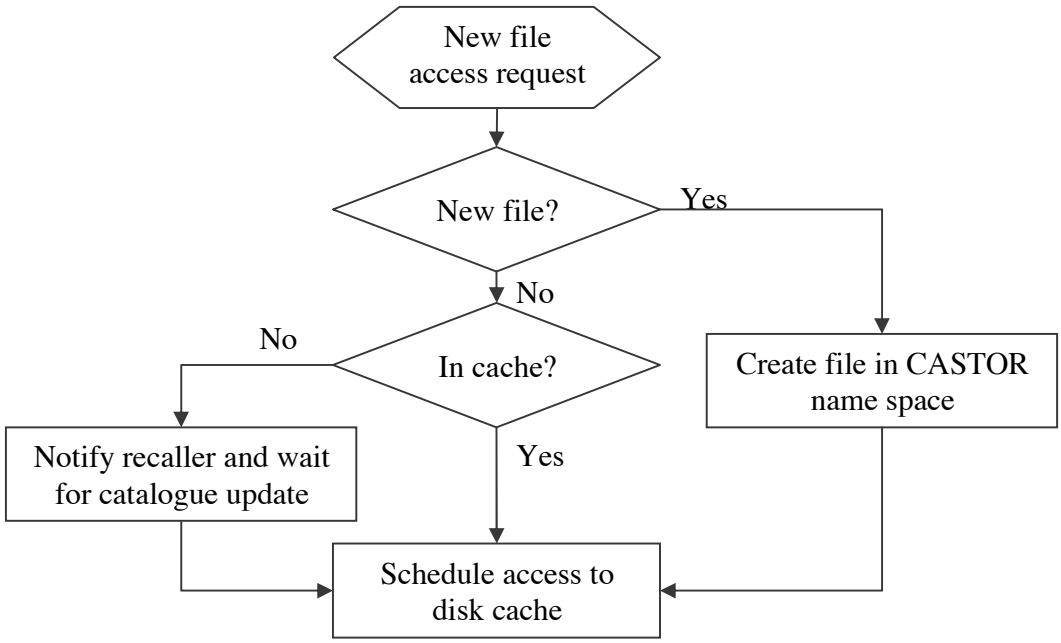


Figure 6: Master stager basic flows

Request throttling and load-balancing are already established concepts in modern disk cache management [2] and distributed file systems [14], [15]. The novelty of the new CASTOR stager architecture is the delegation of the request scheduling to an external module, which allows for leveraging existing and well-tested scheduling systems. In the simplest case, the external scheduling module could be a FIFO queue. However an important advantage of using sophisticated scheduling systems like [7], [8], [16] is that they also provide established procedures for applying policies for quality of service and resource sharing. Interfacing the disk resource management with CPU batch scheduling systems prepares the ground for eventually marrying the CPU and storage resource management.

Because of the close resemblance between farms of CPU servers and farms of commodity file servers it is possible to re-use the CPU batch scheduling systems almost out-of-the-box (see Section 5). The only adaptation required was to accommodate for the fact that a disk cache access request not only targets a file server (node) but also the file system (disk) holding the file or space allocation. While the local pool (or worker directory) file-system is part of normal CPU scheduling attributes, it is normally assumed that there is only one file-system per node. A file server may have several file systems mounted. The CASTOR development team has established excellent collaborations with LSF [7] and Maui [8] developers. Both groups were interested in the problem of disk storage access scheduling and extending their systems to support file-system selection in the scheduling and resource allocation phases.

It should be noted that the similarity between CPU and file server scheduling is partly due to the particular NAS based disk cache architecture with many (100s) file servers holding only a few TB disk space each. For SAN based architectures some more adaptation of the CPU schedulers is probably needed.

4.5. Recaller and migrator components

The recaller and migrator components control the file movements between the CASTOR disk cache and the tape archive. Since most mass storage systems contains similar components, including the current CASTOR stager, only the most important features of the new CASTOR recaller and migrator components will be listed here.

Both the recaller and migrator read the request repository to find out what files needs to be recalled or migrated. While the process of copying files to and from tape is in principle trivial, the task is complicated by the significant delays introduced by tape drive contention and the tape mounting. Tape drive contention comes from the fact that there are usually much more tape volumes than drives and the access must be queued. Tape mounting latency comes from fetching the volume and loading it onto the drive.

Because of the inherent delays introduced by the tape drive contention and tape mounting, CASTOR recaller and migrator do not select the target disk resources that will receive or deliver the data until the tape is ready and positioned. This "deferred allocation" of the disk resources has already been used with good experience in the recaller component of the current CASTOR stager: the target file server and disk that will receive the file is selected in the moment the tape mover is ready to deliver it. In this way space reservations and traffic shaping, which are cumbersome to implement and often lead to suboptimal resource utilization, can be avoided. The selection is currently based on available disk space and I/O load but other metrics/policies can easily be added.

5. First prototype

An evaluation prototype of the new CASTOR stager has been built in order to test out some of the central concepts in the new architecture. The prototype includes:

- A request handler storing all requests in a request repository (Oracle and MySQL supported).
- A master stager reading requests from the request repository and delegates the request scheduling and submission to a LSF instance.
- A LSF scheduler plug-in for selecting target file-system and pass that information to the job when it starts on the file server.
- A load monitoring running on the managed file servers collecting disk load data
- A LSF job-starter that starts an *rfiod* (disk mover) instance for each request scheduled to a file server. The RFIO API client is notified about the *rfiod* address (host and port) and a connection is established for the data transfer.

The prototype was based on standard LSF 5.1 installation without any modifications. This was possible using the plug-in interface provided by the LSF scheduler. A CASTOR plug-in is called for the different phases in the LSF scheduling to perform the matching, sorting and allocation of the file systems based on load monitoring information. While this was sufficient for the prototype, it was recognized that some extra features in the LSF

interfaces would make the processing more efficient. The LSF development team was interested in providing those features and a fruitful collaboration has been established.

The prototype is also prepared for interfacing the Maui scheduler and the Maui developers have been very helpful in providing the necessary interfaces between Maui and the CASTOR file system selection component.

While the prototype was aimed to prove the functionality rather than performance, it should be mentioned that already for this simple deployment using old hardware and without any particular database tuning, the request handling was capable of storing and simultaneously retrieve

- 40 requests/second using Oracle (a dual CPU 1GHz i386 Linux PC with 1GB memory)
- 125 request/second using MySQL (single CPU 2.4GHz i386 Linux PC with 128MB memory)

6. Development status

The developments of the new CASTOR stager started in mid-2003 and a production ready version is planned for 2Q04. Before that a second prototype including all new components and the full database schema is planned for March 2004.

7. Conclusions and outlook

The CASTOR hierarchical storage management system is in production use at CERN since 2001. While the system is expected to scale well to manage the data volumes and rates required for the experiments at the Large Hadron Collider (LHC) in 2007, it has been become clear that the disk cache management will not cope with the expected request load (file opens per second). The development of a new CASTOR stager (disk cache management system) was therefore launched last year. The important features of the new system design are:

- The request processing is shielded from high peak rates by a front-end request handler, which is designed to cope with 500 – 1000 requests per second.
- The scheduling of the access to the disk cache resources is delegated to an external scheduling system. This allows leveraging work and experience from CPU batch scheduling. The first version of the new CASTOR stager will support the LSF and Maui schedulers.
- The software architecture is database centric.

A first prototype of the new system was successfully built in order to prove the new design concepts. The prototype interfaced the LSF 5.1 scheduler.

A production ready version of the new CASTOR stager is planned for 2Q04.

8. References

[1] CASTOR http://cern.ch/castor
[2] dCache http://www.dcache.org/
[3] ENSTORE, http://hppc.fnal.gov/enstore/
[4] HPSS, http://www.sdsc.edu/hpss/

[5] TSM, http://www-306.ibm.com/software/tivoli/products/storage-mgr/
[6] New stager proposal
http://cern.ch/castor/DOCUMENTATION/ARCHITECTURE/NEW/CASTOR-stager-design.htm
[7] Platform computing Inc, http://www.platform.com
[8] Maui scheduler, http://supercluster.org/maui
[9] http://cern.ch/info/Press/PressReleases/Releases2001/PR05.01EUSaward.html
[10] Andrei Maslennikov, New results from CASPUR Storage Lab. Presentation at the HEPiX conference, NIKHEF Amsterdam, 19 – 23 May 2003.
http://www.nikhef.nl/hepix/pres/maslennikov2.ppt
[11] T.Anticic et al, Challenging the challenge: handling data in the Gigabit/s range, Presentation at the CHEP03 conference, La Jolla CA, 24 – 28 March 2003
http://www.slac.stanford.edu/econf/C0303241/proc/papers/MOGT007.PDF
[12] http://cern.ch/info/Press/PressReleases/Releases2003/PR06.03EStoragetek.html
[13] SRM Interface Specification v.2.1, http://sdm.lbl.gov/srm-wg/documents.html
[14] IBM StorageTank,
http://www.almaden.ibm.com/storagesystems/file_systems/storage_tank/index.shtml
[15] Lustre, http://www.lustre.org/
[16] SUN Grid Engine, http://wwws.sun.com/software/gridware/

GUPFS: The Global Unified Parallel File System Project at NERSC*

Greg Butler, Rei Lee, and Mike Welcome
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, California 94720
{GFButler, RCLee, MLWelcome}@lbl.gov
Tel: +1-510-486-4000

Abstract
The Global Unified Parallel File System (GUPFS) project is a five -year project to provide a scalable, high -performance, high -bandwidth, shared file system for the National Energy Research Scientific Computing Center (NERS C). This paper presents the GUPFS testbed configuration, our benchmarking methodology, and some preliminary results.

1 Introduction
The Global Unified Parallel File System (GUPFS) project is a multiple -phase, five year project to provide a scalable, high -performance, high-bandwidth, shared file system for the National Energy Research Scientific Computing Center (NERSC) [1]. The primary purpose of the GUPFS project is to make it easier to conduct advanced scientific research using the NERSC systems. This is t o be accomplished through the use of a shared file system providing a unified file namespace, operating on consolidated shared storage that is directly accessed by all the NERSC production computational and support systems.

In order to successfully deploy a scalable high -performance shared file system with consolidated disk storage, three major emerging technologies must be brought together: shared/cluster file systems, cost -effective, high performance Storage Area Networks (SAN) fabrics, and high performa nce storage devices. Although they are evolving rapidly, these emerging technologies are not targeted towards the needs of high performance scientific computing. The GUPFS project is intended to evaluate these emerging technologies to determine the best so lutions for a center -wide shared file system, and to encourage the development of these technologies in directions needed for HPC at NERSC.

The GUPFS project is expected to span five years. During the first three years of the project, NERSC intends to tes t, evaluate, and steer the development of the technologies necessary for the successful deployment of a center -wide shared file system. Provided that an assessment of the technologies is favorable at the end of the first three years, the

* This work was supported by the Director, Office of Science, Office of Advanced Scientific Computer Research, Mathematical, Information, and Computational Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

last two years of the GUPFS project will focus on a staged deployment, leading to full production at the beginning of FY2006.

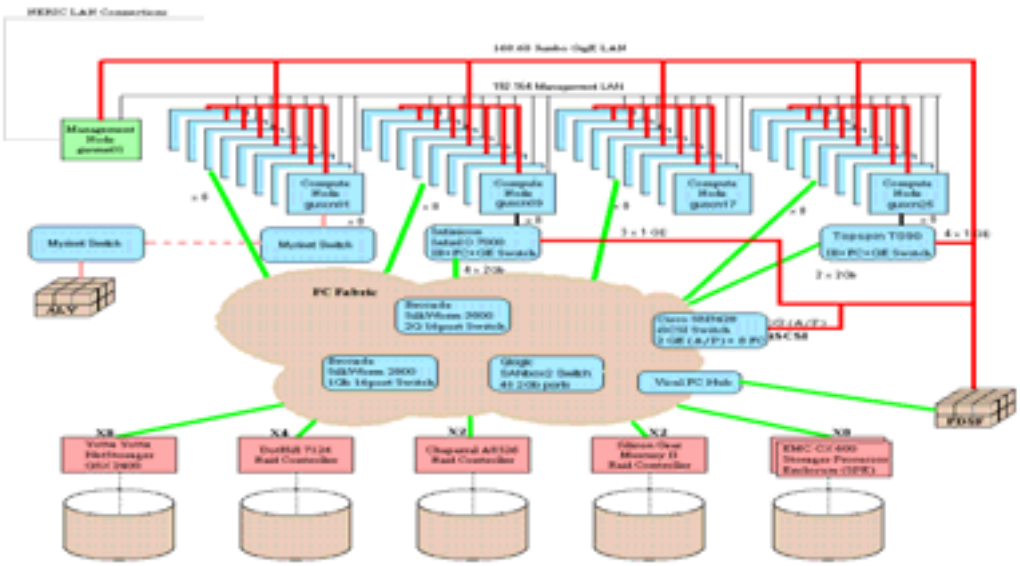
To this end, during the past year the GUPFS project focused on identifying, testing, and evaluating existing and emerging shared and cluster file s ystem, SAN fabric, and storage technologies. During this time, the GUPFS project was also active in identifying NERSC user I/O requirements, methods, and mechanisms, and developing appropriate benchmarking methodologies and benchmark codes for a parallel environment.

This paper presents the GUPFS testbed configuration, our benchmarking methodology, and some preliminary results.

2 GUPFS Testbed Configuration

The GUPFS testbed was constructed from commodity components as a small -scale system mimicking a scien tific parallel computational cluster. Its purpose was to assess the suitability of shared -disk cluster file systems with SAN attached storage to the scientific computational cluster environment.

This testbed system presented a microcosm of a parallel sci entific cluster —dedicated computational nodes, special -function service nodes, and a high -speed interconnect for message passing. It consisted of five computational nodes and one management/interactive node, and utilized an internal jumbo frame Gigabit Eth ernet as the high -speed message passing interconnect. An internal 10/100 Mb/s Fast Ethernet LAN was used for system management and NFS distribution of the user home file systems. The configuration of the testbed is illustrated in following diagram.



In designing a testbed for the GUPFS project, a number of factors were considered. The testbed was designed to support the evaluation of three technology areas:

- Shared/cluster file systems
- SAN fabrics
- Storage devices

These three technology areas are key to the successful deployment of a center-wide shared file system utilizing consolidated storage resources.

The testbed is configured as a Linux parallel scientific cluster, with a management node, a core set of 32 dedicated compute nodes and a set of six special-purpose nodes. Each compute node is a dual Pentium IV system with six PCI-X slots. The PCI-X slots allow us to test newer high performance interfaces such as 4x Infiniband HCA. All computer nodes are equipped with a 2Gb/s Fibre Channel HBA and a 1Gb/s Ethernet interface. Various sets of compute nodes are equipped with different groups of interfaces being evaluated such as Infiniband and Myrinet interfaces.

3 GUPFS Benchmarking Approach

File systems, and parallel file systems in particular, are extremely complicated and should be evaluated within the context of their intended use. In the commercial world, a file system may be evaluated by how it performs on a single application or a small number of critical applications. For example, a web serving content provider may not be interested in parallel write performance since the primary role of the file system is to provide read-only access to data across a large number of servers without having to replicate the data to multiple farms.

By contrast, the NERSC HPC environment must support a large number of user-written applications with varying I/O and metadata performance requirements. Further, applications running today may not resemble the applications that will be running two years from now. Given this diversity, the GUPFS project is taking a more general, multi-pathed approach to the evaluation of parallel file systems.

Initially, the GUPFS project has performed parallel I/O scalability and metadata performance studies. Later, testing includes reliability and stress-test studies, and finally the project will evaluate the performance with respect to specific I/O applications that emulate real NERSC user codes.

3.1 Parallel I/O Performance Studies

Evaluating the performance of a file system, and a parallel file system in particular, can only be done within the context of the underlying system and storage environment. If the underlying storage network or device can only perform at a rate of 30 MB/sec then we cannot expect sustained I/O performance through a file system to exceed this. In addition, for the case of a parallel file system, we need to understand the scalability of the underlying storage network before passing judgment on the file systems' ability to scale to a large number of clients. To aid in this portion of the study, we have developed a parallel I/O benchmark named 'MPTIO' which can perform a variety of I/O operations on files or devices in a flexible manner. MPTIO is an MPI program in which each process spawns multiple threads to perform I/O on the underlying file or device. MPI is

used to synchronize the I/O activity and to collect the per-process results for reporting. It does not use the MPI-I/O library. When acting through a file system, MPTIO can have all threads perform I/O to a single global file, or all the threads of a process can perform I/O to a single file per-process, or all the threads can perform I/O to distinct per-thread files. In addition, I/O can be performed directly to a block device or Linux RAW device to help characterize the underlying storage devices and storage area network. Further, each thread can perform I/O on distinct, non-overlapping regions of the file or device, or multiple threads can perform overlapping I/O. The code can run five different I/O tests, including sequential read and write, random I/O and read-modify-write tests. Aggregate and per-process results are reported. For a complete description of this code, see the *Benchmark Code Descriptions* section in [2].

We can baseline the performance of the storage network and a device using raw device I/O as follows:

- First we measure the I/O rates from a single node to or from the device, by varying the number of I/O threads to the point that the I/O rate saturates.
- Second, we scale up the number of processes (each on a separate node) and also vary the number of I/O threads per process.
- Third, if multiple paths exist through the network to the controller, we can use MPTIO to perform I/O through all paths.

If the raw device I/O rates do not improve past a single node, then the performance bottleneck is in some portion of the network or on the storage device. If they do improve, the first test gives us a good estimation of the peak sustainable raw device I/O rate onto a single node. In this case, as the number of nodes increases, eventually the aggregate raw device I/O rate will again saturate. This bottleneck is either in the network or on the storage controller.

Once a profile of the storage device and network using raw device I/O is complete, a file system can be built over the device. We can then perform I/O and scalability studies over the file system. Since most file systems cache data in host memory, large I/O requests have to be used to minimize the effects of caching. For example, if we observe better performance through the file system than to the RAW device, we can conclude it is the effect of data caching on the node. In addition, one can compare the performance difference between multiple processes writing to the same file versus each writing to different files. If the performance difference is substantial, it will likely be due to write-lock contention between the processes. This might be alleviated if the file system supports byte-range locking so that each process can write its own section without obtaining the single (global) file lock. If the file system supports DIRECT I/O, then one can compare I/O performance through the file system with what was measured to the RAW device. The difference will indicate the amount of software and organizational overhead associated with the file system. For example, file block allocation will probably be distributed across the device resulting in multiple non-contiguous I/O requests to the device. Such I/O performance can degrade as the file system fills and block allocation is more fragmented. File system build options and mount options may also play a

substantial role in the performance. Additional tuning may need to be examined to see how they may affect the I/O performance.

3.2 Metadata Performance Studies

In a typical (local) file system, the metadata is often heavily cached in the system memory and updated in an order such that, in the event of a system crash, the file system would be re-constructible from the on-disk image. Modern file systems maintain transaction logs, or journals, to keep track of which updates have been committed to stable storage and which have not. Each of the data structures generally contain one or more locks such that operating system actors wishing to modify the structure do so in an atomic manner, by first acquiring the lock, modifying the structure and then releasing the lock. Some file system operations require modifying many of the structures and thus the actors must acquire multiple locks to complete the operation. In a parallel file system multiple clients, running on different machines under different instances of an operating system will want to modify these data structures (to perform operations) in an unpredictable order. In these cases, where is the metadata is maintained and who controls it can have a dramatic effect on the performance of a file system operation.

There are currently two main approaches to managing metadata in parallel file systems: symmetric (distributed) and asymmetric. In a purely symmetric file system, all the file system clients hold and share metadata. Clients wanting to perform an operation must request locks from the clients holding them via some from of distributed lock manager. In an asymmetric file system, a central (dedicated) metadata server maintains all the information. The clients request updates or read and write access to files. Clearly, this latter case is easier to manage but establishes a single point of failure and may create a performance bottleneck as the number of clients increases. Note that in both cases, once a client is granted read or write access to a file, it accesses the file data directly through the SAN.

One aspect of evaluating a file system is how well it performs various metadata operations required for concurrent file operations by large number of clients in a parallel environment. Clearly, where and how the metadata is maintained will have a substantial impact on the performance of various file system operations. Clients have to send messages to other nodes in the cluster requesting locks, or asking for operations to be performed. The latency of the interconnection network and the software overhead of processing the communication stack will be a major portion of these costs. Apart from the issues of parallel file systems, a portion of the metadata performance will have to do with how the data structures are organized internally. For example, some file systems will maintain a directory structure as a linear linked list whereas others will use more sophisticated schemes, such as hash tables and balanced trees. Linear lists are easy to implement but access and update performance degrades rather seriously as the number of directory entries increase. The other schemes provide near-constant or log-time access and update rates and perform well as the directory grows. This, of course, is at the expense of a more complicated implementation. Another example is how the file system maintains information about which underlying blocks are free or in use. Some systems use a bitmap whereby the value of the bit indicates the availability of the block. Other

systems use extent-based schemes whereby a small record can represent the availability of a large region of contiguous blocks.

In our study, we measured how the various parallel file systems perform with respect to certain metadata intensive operations. To this end, we have developed a file system metadata benchmark code called 'METABENCH'. This is a parallel application that uses MPI to coordinate processes across multiple file system clients. The processes perform a series of metadata operations, such as file creation, file stating and file utime and append operations. Details on the current state of METABENCH can be found in the *Benchmarking Code Descriptions* section in [2].

3.3 User Applications Emulation

Although micro benchmarks such as MPTIO and METABENCH can provide a wealth of information about how a file system behaves under controlled conditions, the true test of a file system is how it performs in a real user environment. The NERSC user community is large, with a diverse collection of codes that evolve over time. In addition, the codes are complex and may not easily be ported to our test system, or even scale down to that size. Further, the I/O and file operation portion of the code may only consume a small portion of the run-time so attempting to run the actual application on the testbed would be inefficient. In order to address these issues, we plan to develop a small collection of I/O applications that emulate the I/O and file management behavior of real NERSC user applications. This will be a time-consuming task and will only be successful with the aid of the user community. We have begun an informal survey of some of the larger NERSC projects to understand their I/O requirements. As a part of this, we will select a few applications and solicit the users to help us create an I/O benchmark that emulates their code.

4 Preliminary Performance Results

During the past year, we have evaluated a number of products and technologies that we believe are key technologies to the GUPFS Project. We will present testing results in this section for some of the following products and technologies evaluated:

- File Systems: Sistina 5.1 & 5.2 Beta; ADIC StorNext (CVFS) File System 2.0; Lustre 0.6 (1.0 Beta 1) and 1.0; and GPFS 1.3 for Linux
- Fabric Technologies
  - Fibre Channel Switches: Brocade SilkWorm and Qlogic SANbox2-16 and SANbox2-64
  - ISCSI[3]: Cisco SN 5428, Intel iSCSI HBA, iSCSI over IB
  - Infiniband: InfiniCon and Topspin (IB to FC and GigE)
  - Inter-connect: Myrinnet, GigE
- Fibre Channel Storage Devices
  - 1Gb/s FC: Dot Hill, Silicon Gear, Chaparral
  - 2Gb/s FC: Yotta Yotta NetStorager[8], EMC CX 600, 3PARdata

4.1 Storage Performance and Scalability

Storage can be a performance bottleneck of any file system. A storage device may be able to sustain a very good single -port performance. However, having good single -port performance is not sufficient for a shared disk file system like GUPFS. For GUPFS, the underlying storage devices must demonstrate a very good scalability wh en the number of clients increases (to thousands or tens of thousands). A shared file system will not scale if the underlying storage does not scale.

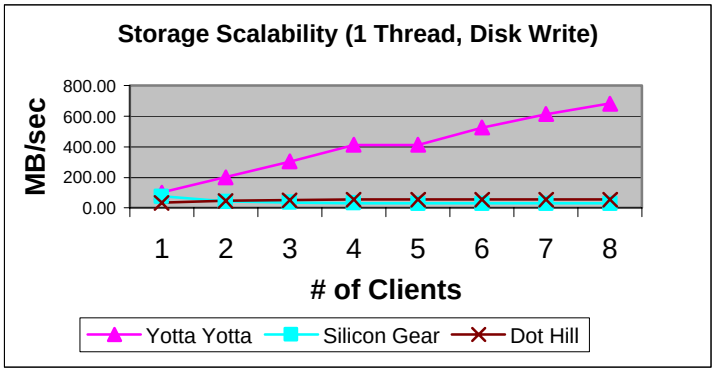


Figure 1. Storage Scalability

Figure 1 shows how st orage devices scale when the number of clients increase. The figure shows the results of three storage devices: Yotta Yotta GSX 2400 (YY), Silicon Gear Mercury II (SG), and DotHill SANnet (DH). Both Silicon Gear and Dot Hill have only 2 1Gb/s front -end ports, while Yotta Yotta has 8 2Gb/s ports and 3PARdata has 16 2Gb/s ports but only 8 were used during the test. On each storage device, we created a single LUN to be shared by multiple clients for shared access.

The figure shows that both the Silicon Gear and Dot Hill devices did not scale when the number of clients increased. Silicon Gear performance actually dropped when the number of clients increased. On the other hand, the figure shows that the Yotta Yotta storage did scale very well when the number of clients increased.

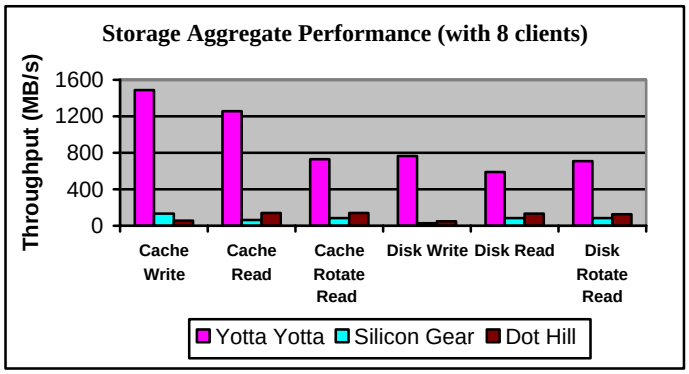


Figure 2. Storage Aggregate Performance

Figure 2 shows the aggregate performance of the three storage devices: DotHill, Silicon Gear, and Yotta Yotta, using the MPTIO benchmark w ith different test conditions. The

results indicate that the Yotta Yotta storage will be able to sustain higher performance than Silicon Gear or Dot Hill in a shared file system.

4.2 Parallel File I/O Performance

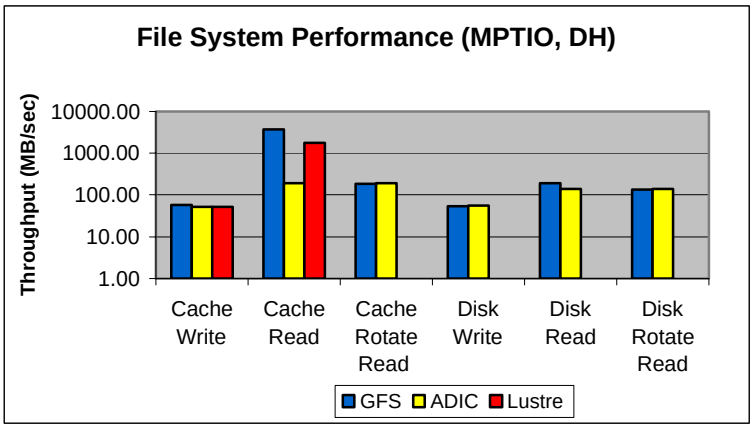


Figure 3. Shared File System Performance

During the last year, we have tested several file systems, including Sistina's GFS [4], ADIC's StorNext File System [5], and Lustre [7]. The above diagram shows the 8 -client MPTIO results on these file system, under different test scenarios. These results indicate that there is not much difference in parallel I/O performance between GFS and ADIC's StorNext File System, except for 'Cache Read'. ADIC's StorNext File System was probably doing direct I/O even when operating on files that can fit in the OS cache.

With the award of the ASCI PathForward SGSFS [6] file system development contract to HP and Cluster File Systems, Inc., there has been rapid progress on the Lustre file system [7]. The earlier Lustre file system version (0.6) we tested failed to complete all but the 'Cache Write' and 'Cache Read' tests. Luster1.0.0 was recently released and Figure 4 shows the latest result of Lustre scalability with six clients and two Object Storage Servers (OSS).

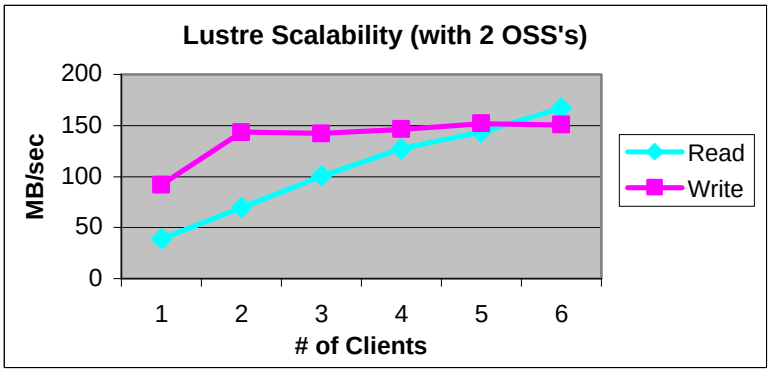


Figure 4. Lustre Scalability

The figure seems to indicate that reads and writes were limited by the GigE interface as the test was running with two OSS's and each OSS was equipped with one GigE interface. Additional test with more OSS's and tuning should improve performance.

4.3 Fabric Performance

Storage area networks, by providing a high performance network fabric oriented toward storage device transfer protocols, allow direct physical data transfers between hosts and storage devices. Currently, most SANs are i mplemented using Fibre Channel (FC) protocol-based fabric. Emerging alternative SAN protocols, such as iSCSI (Internet Small Computer System Interface), FCIP (Fibre Channel over IP), and SRP (SCSI RDMA [Remote Direct Memory Access] Protocol) [9], are enabl ing the use of alternative fabric technologies, such as Gigabit Ethernet and the emerging InfiniBand, as SAN fabrics.

Here we present some performance results of several fabric technologies: Fibre Channel (FC), iSCSI over GigE, iSCSI over IP over Infiniband (IPoIB), and SRP.

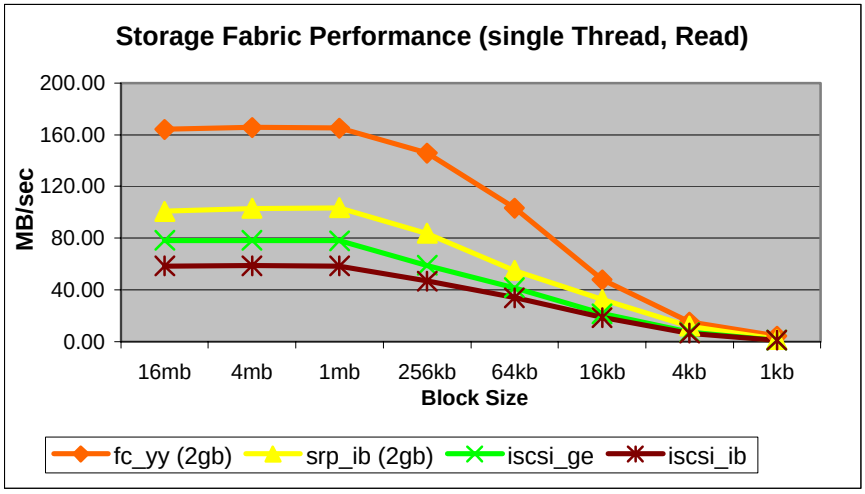


Figure 5. Storage Fabric Performance

Figure 5 shows the results of single -thread reads of different I/O size using different fabric technologies. The best performance was achieved by the 2Gb/s FC interfa ce, followed by the SRP protocol over Infiniband. Since the iSCSI traffic was passing through a single GigE interface, the iSCSI performance was less than 100 MB/s. With the additional stack overhead of IPoIB, iSCSI over IPoIB delivered the lowest performa nce for single-thread reads.

Figure 6 shows the CPU overhead of different protocols for single-thread reads. FC, while delivered the best performance, used the least CPU overhead. The iSCSI protocol allows the standard SCSI packets to be enveloped in IP packets and transported over standard Ethernet infrastructure, which allows SANs to be deployed on IP networks. This option is very attractive as it allows lower-cost SAN connectivity than can be achieved with Fibre Channel, although with lower performance. It will allow large numbers of inexpensive systems to be connected to the SAN and use the shared file system through

commodity-priced components. While attractive from a hardware cost perspective, this option does incur a performance impact on each host due to increased traffic through the host's IP stack, as shown in Figure 6.

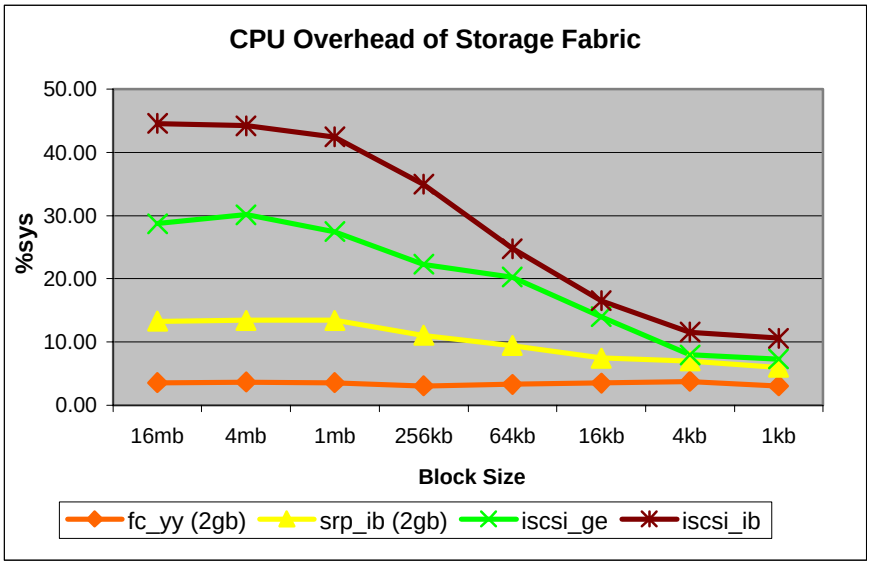


Figure 6. CPU Overhead of Storage Fabric

5 Conclusions

The GUPFS project started in the last half of FY 2001 as a limited investigation of the suitability of shared-disk file systems in a SAN environment for scientific clusters, with an eye towards possible future center-wide deployment. As such, it was targeted towards initial testing of the Sistina Global File System (GFS), and included a small testbed system to be used in the investigation.

With the advent of the NERSC Strategic Proposal for FY 2002–2006, this modest investigation evolved into the GUPFS project, which is one of the major programmatic thrusts at NERSC. During the first three years of the GUPFS project, NERSC intends to test, evaluate, and influence the development of the technologies necessary for the successful deployment of a center-wide shared file system. Provided that an assessment of the technologies is favorable at the end of the first three years, the last two years of the GUPFS project will focus on a staged deployment of a high performance shared file system center-wide at NERSC, in conjunction with the consolidation of user disk storage, leading to production in FY 2006.

Reference

[1] NERSC Strategic Proposal FY2002-FY2006, http://www.nersc.gov/aboutnersc/pubs/Strategic_Proposal_final.pdf.
[2] The Global Unified Parallel File System (GUPFS) Project: FY 2002 Activities and Results, http://www.nersc.gov/aboutnersc/pubs/GUPFS_02.pdf.
[3] Julian Satran, "iSCSI" (Internet Small Computer System Interface) IETF Standard, January 24, 2003, http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.pdf.

[4] Global File System (GFS), Sistina Software, Inc., http://www.sistina.com/downloads/datasheets/GFS_datasheet.pdf.
[5] StorNext File System, Advanced Digital Information Corporation (ADIC), http://www.adic.com/ibeCCtpSctDspRte.jsp?minisite=10000&respid=22372&section=10121.
[6] ASCI Path Forward, SGSFS, 2001, http://www.lustre.org/docs/SGSRFP.pdf.
[7] "Lustre: A Scalable, High-Performance File System," Cluster File Systems, Inc., November 2002, http://www.lustre.org/docs/whitepaper.pdf.
[8] Yotta Yotta NetStorager GSX 2400, Yotta Yotta, Inc., http://www.yottayotta.com/pages/products/overview.htm.
[9] *SRP:* SCSI RDMA Protocol: ftp://ftp.t10.org/t10/drafts/srp/srp-r16a.pdf.

SANSIM: A PLATFORM FOR SIMULATION AND DESIGN OF A STORAGE AREA NETWORK

Yao-Long Zhu, Chao-Yang Wang, Wei-Ya Xi, and Feng Zhou
Data Storage Institute
DSI building, 5 Engineering Drive 1, (off Kent Ridge Crescent, NUS)
Singapore 117608
Tel: +65-6874-6436
e-mail: zhu_yaolong@dsi.a-star.edu.sg

Abstract

Modeling and simulation are flexible and effective tools to design and evaluate the performance of Storage Area Network (SAN). Fibre Channel (FC) is presently the dominant protocol used in SAN. In this paper, we present a new simulation - SANSim, developed for modeling and analyzing FC storage network. SANSim includes four main modules: an I/O workload module, a host module, a storage network module, and a storage system module. SANSim has been validated by comparing the simulation results with the actual I/O performance of a FC RAM disk connected to a FC network. The simulated results match the experimental readings within 3%. As an example of applicability, SANSim has been used to study the impact of link failures on the performance of a FC network with a core/edge topology.

1. Introduction

SAN architecture has been proven to provide significant performance advantages, larger scalability, and higher availability over traditional Direct Attached Storage architecture. It is therefore not surprising that the performance modeling and simulation of SANs has become an interesting field of research [1][2]. Xavier [3][4] used the CSIM language to simulate a SAN and model its activities. Petra et al. [5] presented the SIMLab as a simulation environment based on a network of active routers. Wilkes [6] used the Pantheon storage-system simulator to model the performance of parallel disk arrays and parallel computers. DiskSim [7] is another disk storage system simulator to support research in storage subsystem algorithm and architecture.

However, the simulation studies and tools mentioned above have been very limited in modeling and simulation at the FC protocol level. Nevertheless, it is necessary to simulate at the frame-level FC in order to monitor and analyze details of FC SAN activities.

In this paper, we present a new FC SAN simulation, SANSim, which supports FC frame-level and fully simulates Fibre Channel protocols in accordance to the relevant standards [10-13] to guarantee the compatibility and interoperability of different modules. In the following sections, we first introduce our simulation tool SANSim in section 2. Then, we present in section 3, the experimental results and simulation validation of a FC network. Finally, we simulate and analyze some FC network design issues in section 4.

2. SANSim

SANSim is an event-driven simulation tool for SAN that includes four main modules: an I/O workload module, a host module, a storage network module, and a storage system module, as shown in Figure 1.

The I/O workload module generates I/O request streams according to the workload distribution characteristics and sends them to the host modules. The host module encapsulates the I/O workload to the SCSI commands and sends them to the Host Bus Adaptor (HBA) sub-modules. The storage network module simulates the network connectivity, topology and communication mechanism. The FC network module includes three sub-modules: a FC_controller module, a FC_switch module and a FC communication module. The storage module maps I/O data to the storage devices.

SANSim is developed in pure standard C. It has been compiled successfully both in Microsoft Window XP and Linux platforms. The simulator reads in configuration parameters from a user specified input file, plots measurement data, and writes data in an output file after the simulation is completed. The simulation duration and the warm-up period can also be specified in the input file to control and eliminate the transient bias in the simulation results. The configuration parameters for each of the four modules are

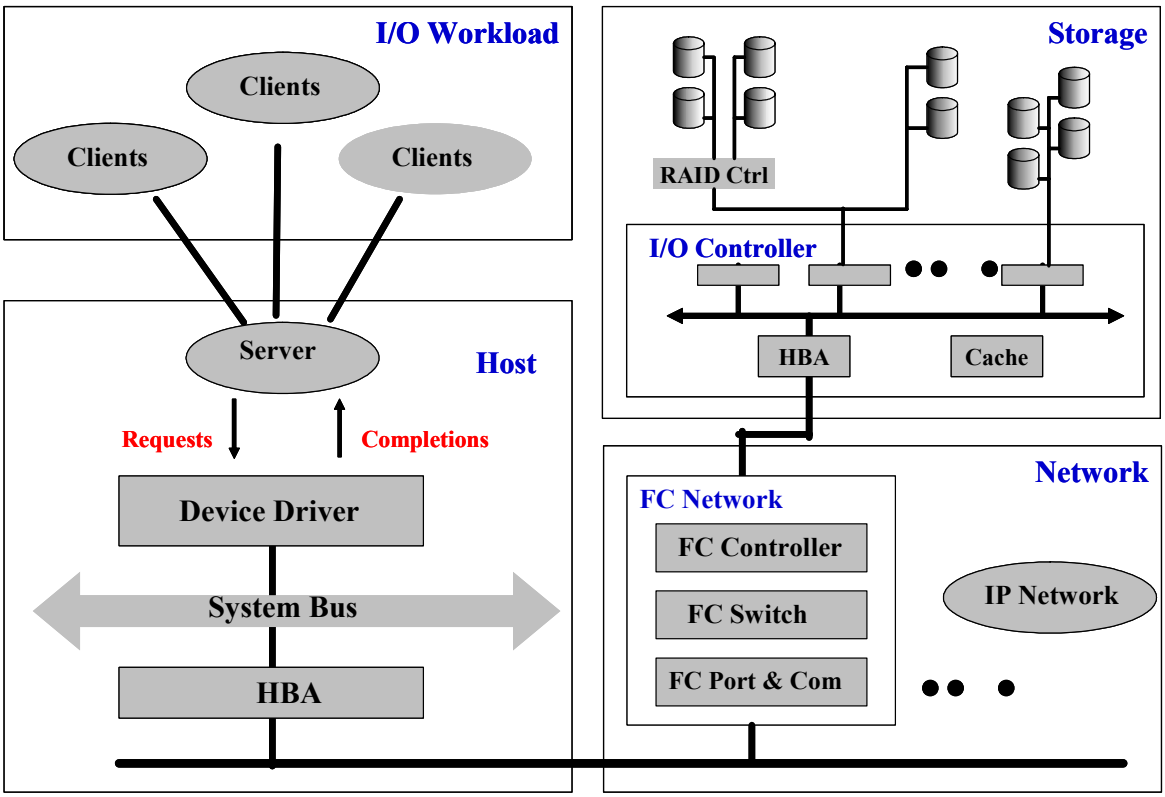


Figure 1 SANSim internal structure including I/O workload module, host module, storage network module and storage module.

arranged orderly in the input file.

2.1 I/O Workload module

The key function of I/O Workload module is to generate I/O request streams according to the workload distribution characteristics and send them to the host modules. The current module supports both system-traced I/O workloads and synthetic I/O workloads. We use a synthetic I/O workload in this paper.

Generally speaking, a disk request is defined by five dimensions: the requested pattern, the size distribution, the repeatability, the location distribution and the I/O operations. The workload module is able to generate several basic different arrival patterns such as Poisson arrivals, equal time intervals, Normal distribution arrivals and so on. It can also simulate a combination of request patterns that consists of the different distributions and different rates. Another capability of the Workload module is to generate repeatable requests. This scenario is used to define a workload in which some files are more popular than others and consequently accessed more frequently.

2.2 Host module

Host module includes a device driver, a SCSI layer, a system bus, as well as DMA modules. The main function of the Host module is to encapsulate the I/O workload into the SCSI commands and sends them to the Host Bus Adaptor (HBA) sub-modules.

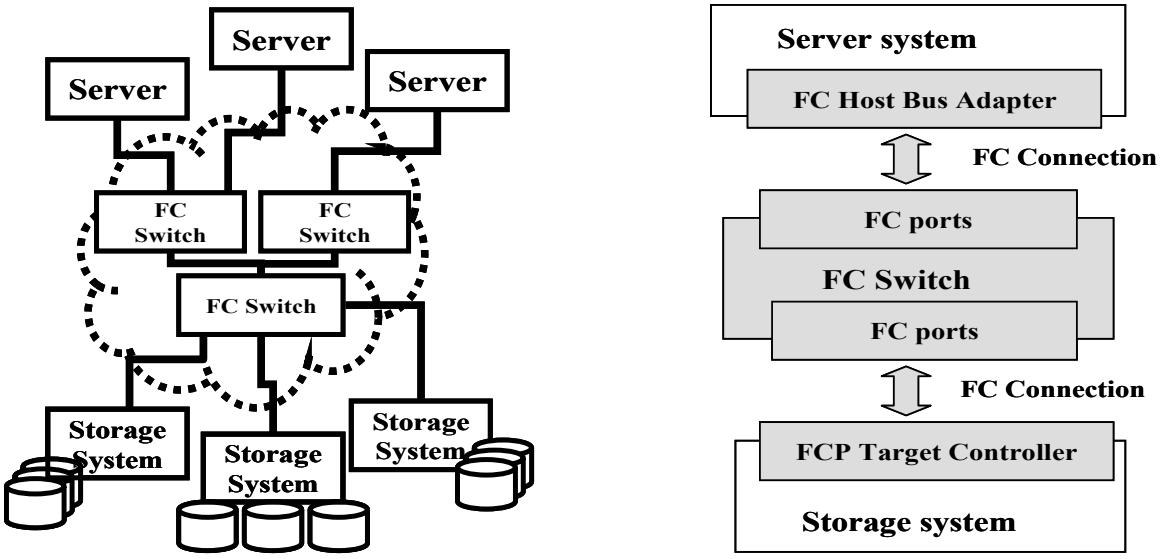
The host module schedules the I/O requests generated by the workload module, based on various schedule policies, which are configurable. The I/O requests are traced in a waiting queue and an outstanding queue. An I/O request in the outstanding queue refers to a request, which has been submitted to the storage device, but has not been completed yet. The maximum number of outstanding requests depends on the buffer size and system configuration. The I/O requests in the waiting queue may be merged or re-scheduled following certain policies.

The host module supports a multiple-host configuration. Each host has a separate I/O generation module. There is a specific mechanism to identify the I/O requests coming from different hosts.

2.3 FC network module

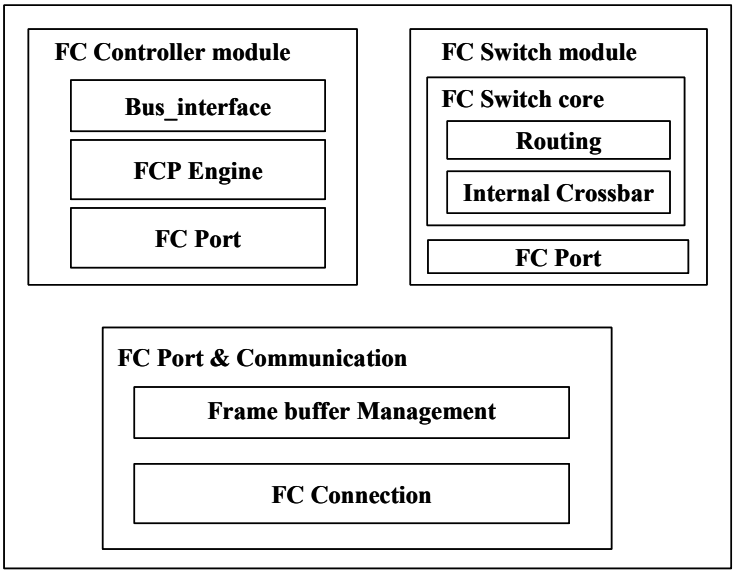
The key function of the FC network module is to simulate the FC connectivity, topology and communication protocol. The FC network module includes three sub-modules: the FC Controller module, the FC Switch module and the FC Port & Communication module, as shown in Figure 2. The FC Controller module simulates the communication behavior of FC Commands or Data Frames. The FC Switch module models all the FC Ports, switch architecture, and as well as the routing and flow control. The FC Port & Communication module transfers FC Frames between the FC Ports.

2.3.1 FC Controller module



(a) An example of FC SAN

(b) Abstracted view



(c) FC network module

Figure 2 Modeling of Fibre Channel network in SANSim

The FC Controller module models both initiator and target modes of the FC HBA. As shown in Figure 3, the FC Controller module includes three sub-modules: a Bus_interface, an FCP(SCSI over Firbre Channel Protocol [13]) Engine and a FC Port. The Bus_Interface sub-module handles the communication between the device driver and the controller such as DMA and interruptions. The FCP Engine has the responsibility of

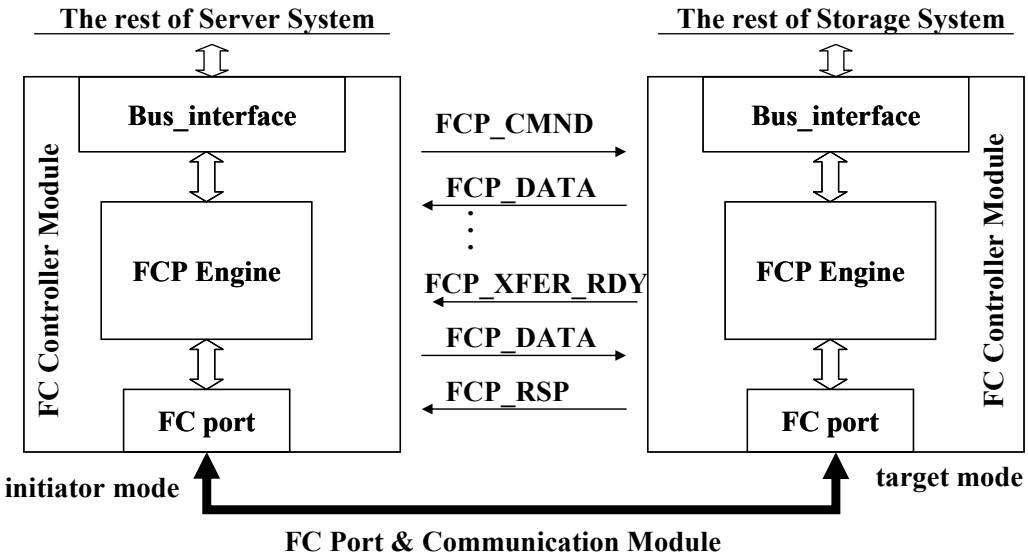


Figure 3 FCP Operation

constructing different FC frames corresponding to each sequence of FCP exchange. The FC_Port is responsible for delivering FC frames to the destination port.

When a SCSI request arrives in the initiator, the device driver sends SCSI commands to the FC_Controller through the Bus_Interface. The FC_Controller then executes the commands and fetches SCSI I/O's information from memory. A FCP_CMND frame is constructed for each SCSI I/O in the FCP Engine, and then sent out through FC_Port module. After the target FC_Controller receives the FCP_CMND, the target firmware decapsulates the FCP_CMND and processes the SCSI command.

In the case of a FCP Read operation (SCSI Read 10), the target host decapsulates the SCSI request and prepares the requested data. Once the data is ready, the target FC_Controller takes the responsibility of transferring data back to the requestor (the initiator) and sends a completion message FCP_RSP after all data is sent. With a FCP Write operation (SCSI Write 10), the target driver allocates sufficient memory area for the incoming data, and sends a FCP_XFER_RDY to the initiator requesting the data. After the initiator receives the FCP_XFER_RDY, it starts to send a FCP_DATA. Finally when the target receives all data successfully, it sends a FCP_RSP indicating the completion of the FCP operation.

2.3.2 FC Switch module

SANSim FC Switch module has two sub modules: FC port, and FC Switch core. FC port supports F_Ports/FL_Ports and E_Ports. F_Ports/FL_Ports are for host-switch and device-switch connections, and E_Ports are for switch-switch inter-connections. The FC_Port's address_ID is unique and confined to the FC-SW-2 standard [11]. FC Switch core is the switch's control center for frames routing and forwarding. It contains routing and internal cross-bar. If the destination port of requested FC frames is busy, the incom-

ing frames are held in the incoming buffer until they are successfully routed. SANSim uses Dijkstra's algorithm [14] to compute the shortest routing path. The routing table remains constant unless the network connectivity is changed during the simulation. When the network configuration is changed, the switch module re-computes the shortest path.

2.3.3 FC Port & Communication module

The FC Port & Communication module includes a Frame Buffer management and FC connection sub modules. The Frame Buffer management sub module handles all management of incoming and outgoing frame buffers. The FC connection sub module establishes a FC connection between two FC_Ports used to transfer frames.

The Fibre Channel Arbitrated Loop (FC-AL) is a typical sub FC connection sub module used in the simulation. The FC-AL connection module consists of the following sub modules: Loop Port State Machine (LPSM), Alternative BB Credit Flow Control (Alt BB Mgr), and Loop Port Process Control (LPPC). The LPSM models the L_Port's state transition during the communication. The L_Port transmits FC Frame only when it is in certain states. The number of frames permitted to be transmitted depends on the available buffer size in the destination port. The Alternative BB Credit Flow Control sub module models the flow control method defined in the Standard [10] to avoid overflow. The Loop Port Process Control sub module models the remaining behavior of the L_Port, such as responding to a certain loop port request, re-transmitting an ARBx signal and other port activities.

2.4 Storage module

The main function of the storage module is to map I/O data to the storage devices. Storage modules, which include interface module (HBA), storage controller module and storage device module, can simulate a RAID system with various cache management algorithms, disk drives, and RAM disks. In the event of disk failure in a RAID system, the degraded mode and rebuild behavior can also be modeled. Various RAID algorithms can be integrated into the storage modules.

3. Simulation Validation

3.1 Experimental environment

The experiments were conducted on an in-house developed FC RAM disk, which maps all storage I/Os to the memory rather than using an actual magnetic disk. Since we are focusing on the FC simulation and validation, using RAM disk as a target helps to isolate problems caused by the modeling of a hard disk drive. The initiator and target use a FC-AL connection. Table 1 lists the detailed hardware and software configurations used in the experiments. I/OMeter, a widely accepted industry standard benchmark tool, is used here to collect experimental data. The monitored parameters include IOPS (I/O per second), throughput (MB/s), queue depth, I/O request size, and read/write operations. The IOPS is used primarily with small I/O requests, while the throughput is used with large I/O requests. The queue depth refers to the number of outstanding I/O requests, which are injected into the FC network and storage system. The I/OMeter issues a number of re-

Table 1 System configuration for SANSim FC-AL module validation

Initiator	
Hardware	CPU: AMD AthonMP 1600+ FC HBA: Qlogic 2300 RAM: 2x256MB DDR SDRAM Main board: 64 bit PCI Tyan Tiger MP2466N
Software	OS: Windows XP Professional SP1 Driver: Qlogic Driver Version 8.1.5.12 Tool: Intel IOMeter Version 2003.02.15
Target	
Hardware	CPU: Intel PIII 1GHz FC HBA: Qlogic 2300 RAM: 4 x 1GB Kingston ECC Reg. PC133 Main board: 64bit PCI, Supermicro 370
Software	OS: RedHat 8.0 Kernel: 2.4.18 Driver: In-house 2300 target driver Version 1.0, In-house Linux RAM Disk Version 2.0

quests (equals to the queue depth) initially, and generates new I/O requests only after the completion of previous requests. Fixed I/O sizes were used in all requests.

3.2 Comparisons of the experimental and simulated data

Figure 4 and 5 show that I/O transaction performance varies with the queue depth with read and write operations. The I/O sizes are set to 2KB, 8KB, 16KB, and 32KB respectively. The IOPS increases with the queue depth and then reaches a saturation limit. In other words, there is a critical queue depth. When the queue depth is bigger than this critical value, the system transaction performance shows no improvement and the response time for I/O request becomes worse. For example, when an I/O size is 8KB and IOMeter sends 100% of read operations, the maximum transaction performance is 17.5k IOPS. This translates into a network and storage overhead means of around 0.057ms (1second/17.5k). Simulation results show that the average response time is about 0.471ms when queue depth equals to eight. The largest contribution in the response time is from the queue waiting time, not the serving time.

Figure 6 and 7 show that the throughput (MB/s) varies with the I/O request size with read/write operations. The queue depths are set to 1, 2, and 8 respectively. Generally, the throughput increases with I/O request sizes. When the request size is large enough (more than 128KB with queue depth of 8), the data transfer time dominates the overall overhead. Then the throughput is limited by the FC network bandwidth.

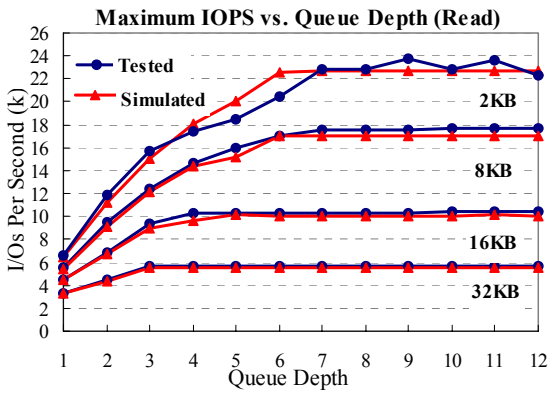


Figure 4 Read performance vs. queue depth

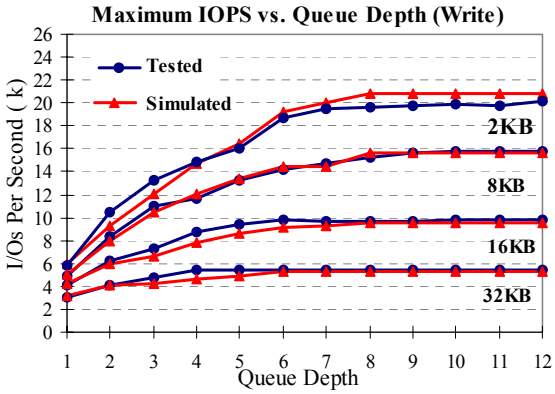


Figure 5 Write performance vs. queue depth

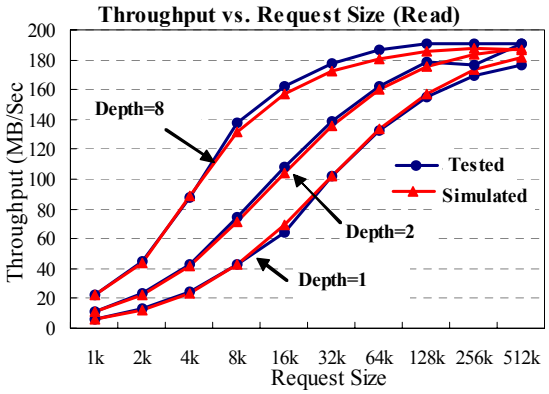


Figure 6 Read throughput vs. I/O size

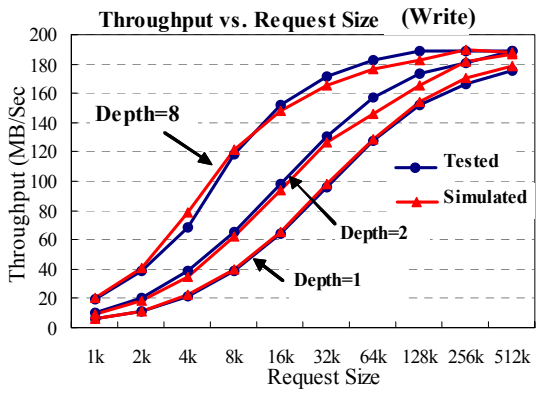


Figure 7 Write throughput vs. I/O size

The simulation results and experimental data match very well in all cases as illustrated in figures 4-7. The error range is generally less than 10%. With read operations, it is less than 3%.

4. FC Network simulation and analysis

4.1 Simulation Environment

To illustrate an application of SANsim, the performance and availability of a FC network are simulated and analyzed in this section. We conducted a simulation based on the core/edge network with five FC switches, as shown in Figure 8. FC switches 1-4, as edge switches, are connected to the core switch 5 through two 2G FC ISLs. Each FC controller has 32 frame buffers. Each FC port on a switch has four frame buffers. The distance between any two points in the network is set to be 50 meters. The storage controller processing capacity is 22.5K transactions per second.

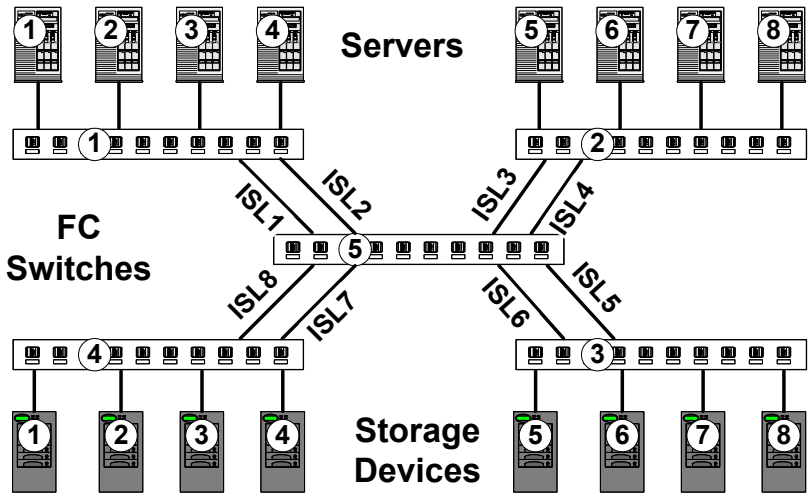


Figure 8 Simulation configuration

A synthetic I/O workload is applied to each server. The I/O inter-arrival time follows an exponential distribution. The maximum number of outstanding I/O requests is 32 for each server. Since the simulation is based on an open system, the I/O queue in the server is allowed to grow without limitation. However, the servers issue a new request to the network and storage devices only after a previous request is completed. The IO requests are randomly and evenly allocated among all devices.

4.2 Simulation Results and Analysis

In order to study the impact of link failure on the network throughput, we conducted a series of simulations under four different scenarios: no link failure, ISL1 failure, ISL8 failure and simultaneous failures of ISL1 and ISL8. The throughputs of all servers are collected using 2KB and 32 KB request sizes. Servers 1 through 4 have the same characteristics and achieve similar throughput results. We use an average throughput S1-4, as shown in Figure 9, to represent the performance of Server 1 though 4. Same process is applied to server 5 through 8.

Figure 9(a) shows the throughputs as a function of I/O workload for case I (without link failure). The throughputs grow linearly and then reach asymptotic values. All servers achieve the same throughputs due to the symmetrical characteristics of the network. When the I/O size is 2KB, the maximum throughput is 45MB/s. Since the process capacity of the storage device is 22.5k transactions per second, it limits the maximum throughput to 45MB/s for 2KB I/O size. When the I/O size is 32KB, the maximum throughput for each server is 80MB/s. The total throughput for a single network link is 160MB/s and that is 20% less than the nominal value 200MB/s. The simulated results show that the maximum throughout supported by a single storage device is 175MB/s for 32KB I/O size. That means the performance is not limited by the storage devices, but by the network.

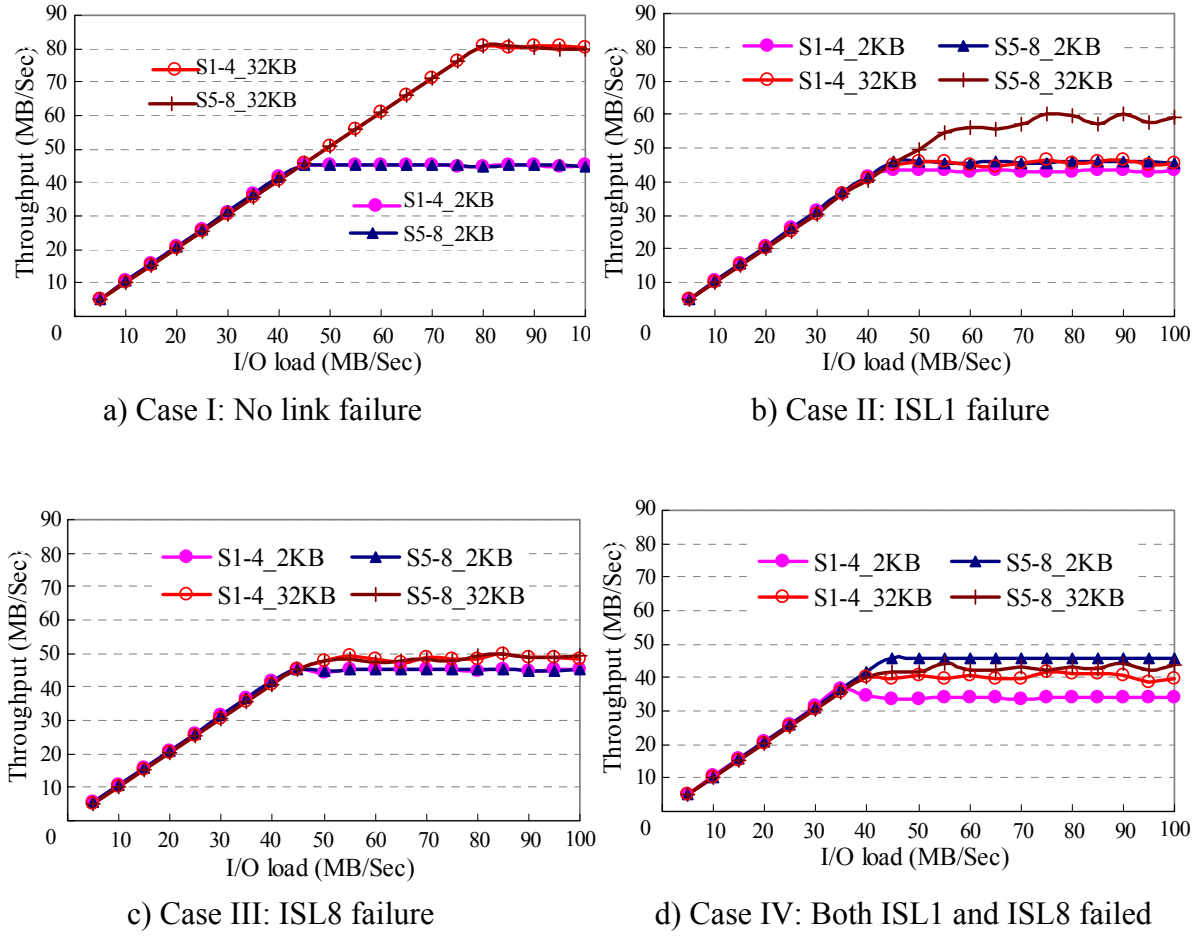


Figure 9 Maximum throughputs under symmetrical I/O load for different cases

Figure 9(b) shows the throughput for the case II (with ISL1 failure). When the I/O size is 32KB, the asymptotic throughput of servers 1~4 drops to 46MB/s, compared to 80MB/s in case I. Apparently, this is due to the limited bandwidth of a single link ISL2. It is also noted that a large performance drop happens with servers 5~8 (58MB/s vs. 80MB/s) compared to case I, even though the ISL1 has no direct physical connection to those servers. This decrease in performance is probably caused by the head-of-line blocking [15]. The data traffics from storage devices to servers 5~8 are competing at the core switch 5 with the data traffics from storage devices to servers 1~4. When the I/O size is 2KB, the throughput of servers 1~4 approaches to 44MB/s which is slightly less than the 45MB/s in the case I. This is caused by the competition of data traffic on the highly utilized ISL2 link. The bandwidth utilization of ISL2 reaches 176MB/s over 200MB/s.

For case III (with ISL8 failure), the maximum throughputs for servers 1~4 and servers 5~8 drop from 80MBs in case I to an average of around 50MB/s, as shown in Figure 9(c), when the I/O size is 32KB. The reason why servers 1~4 and servers 5~8 achieve the same maximum throughput is because the data traffics from storage devices 1~4 to servers 1~8

are equally affected by ISL8 failure. When the I/O size is 2KB, the maximum throughput is almost not affected by the ISL8 failure compared to case I.

When both ISL1 and ISL8 fail (case IV), the measured throughputs of servers are shown in Figure 9 (d). When I/O request size is 32KB, the asymptotic throughput of server 1-4 reaches 40 MB/s while the throughput of servers 5-8 is about 42MB/s. However, when the I/O size is 2KB, the throughput of servers 1~4 is only 35MB/s, and servers 5~8 is 46MB/s. In order to analyze the detailed frame activity on the network, data traffic across ISL1~4 and ISL5~8 are monitored and the average I/O response time are measured. The results show that the response time of the I/O issued by servers 1~4 to storage devices 1~8 becomes significant long (>2ms) when the I/O workload is larger than 36.5MB/s. This allows the storage devices to serve more I/O requests issued by servers 5~8. So servers 5~8 can achieve better performance than servers 1~4 when the I/O size is 2KB. However, when the I/O size is 32KB, the response time of I/O requests issued by servers 5~8, increases notably due to the effects of header-of-line blocking. It limits the performance of the servers 5~8 to 42MB/s with 32 KB I/O size.

5. Summary and future work

In this paper, we have presented SANSim, a platform for simulation and design of a FC SAN. The SANSim, which is based on FC frame level, can simulate all primitive signals (IDLEs, RDYs etc), commands and data frames. The design of SANSim is modular and scalable. Such tool is useful to the rapid development of high-end SAN due to the ever-increasing complexity of the SAN architecture.

We have conducted several experiments to compare experimental and simulated results. The results show that SANSim model is accurate within less than 3% in read operation, and less than 10% in write operation. As an example, the performance and availability of a core/edge FC network has been analyzed. The simulation results show that the core/edge topology suffers from certain level of bandwidth loss due to the Head-of-Line blocking caused by traffics crossing multi-stage switches. Generally, the maximum throughput achieved at all servers decreases when link failure happens. The servers on different locations have different I/O performance sensitivity to the link failure.

Future development work of SANSim includes IP storage module, Object-based storage module, file system simulation.

Reference

[1] Yao-Long Zhu, Shun-Yu Zhu and Hui Xiong, " Performance Analysis and Testing of the Storage Area Network", 19th IEEE Symposium on Mass Storage Systems and Technologies, April 2002.

[2] T. Ruwart, "Disk Subsystem Performance Evaluation: From Disk Drivers To Storage Area Networks", 18th IEEE Symposium on Mass Storage Systems and Technologies, April 2001.

[3] Xavier Molero, Federico Silla, Vicente Santonja and José Duato, "Modeling and Simulation of Storage Area Networks", Modeling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE 2000.

[4] Xavier Molero, Federico Silla, Vicente Santonja and José Duato, "A Tool For The Design And Evaluation Of Fibre Channel Storage Area Networks", Proceedings of 34th Simulation Symposium, 2001.
[5] Petra Berenbrink, André Brinkmann and Christian Scheideler, "SIMLAB - A Simulation Environment for Storage Area Networks", 9th Euromicro Workshop on Parallel and Distributed Processing (PDP), 2000.
[6] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan, "The HP AutoRAID Hierarchical Storage System", ACM Transactions on Computer Systems, 1996.
[7] Gregory R. Ganger and Yale N. Patt., "Using System-Level Models to Evaluate I/O Subsystem Designs", IEEE Transactions on Computers 1998.
[8] Thomas M. Ruwart, "Performance Characterization of Large and Long Fibre Channel Arbitrated Loops", IEEE Network 1999.
[9] John R. Heath and peter J. Yakutis, "High-Speed Storage Area networks Using Fibre Channel Arbitrated Loop Interconnect", IEEE Network 2000.
[10] FC-AL, "FC Arbitrated Loop," ANSI X3.272:1996
[11] FC-PH, "Fibre Channel Physical and Signaling Interface (FC-PH)", ANSI X3.230:1994.
[12] FC-SW, "FC Switch Fabric and Switch Control Requirements", ANSI X3.950:1998.
[13] Technical Committee T11, FC Projects http://www.t11.org/Index.html.
[14] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", Numerische Mathmatik 1 (1959) 269-271
[15] M. Jurczyk, "Performance and Implementation Aspects of Higher Order Head-of-Line Blocking Switch Boxes", Proceedings of the 1997 International Conference on Parallel Processing, IEEE 1997

Cost-Effective Remote Mirroring Using the iSCSI Protocol

Ming Zhang, Yinan Liu, and Qing (Ken) Yang
Department of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI 02874
{mingz, yinan, qyang}@ele.uri.edu
tel +1-401-874-5880
fax +1-401-782-6422

Abstract

This paper presents a performance study of the iSCSI protocol in the context of remote mirroring. We first integrate our caching technology called DCD (disk caching disk) into a standard iSCSI target device to form a high performance storage system for mirroring purpose. Performance measurements are then carried out using this storage system as well as standard iSCSI targets as mirroring devices. We consider remote mirroring on a LAN (local area network) and on a commercial WAN (wide area network). The workloads used in our measurements include popular benchmarks such as PostMark and IoMeter, and real-world I/O traces. Our measurement results show that iSCSI is a viable approach to cost-effective remote mirroring for organizations that have moderate amount of data changes. In particular, our DCD-enhanced iSCSI target can greatly improve performance of remote mirroring.

1. Introduction

Remote data mirroring has become increasingly important as organizations and businesses depend more and more on digital information [1]. It has been widely deployed in financial industry and other businesses for tolerating failures and disaster recovery. Traditionally, such remote mirroring is done through dedicated SAN (storage area network) with FC (Fiber Channel) connections that are usually very costly in terms of installation and maintenance. A newly emerging protocol for storage networking, iSCSI [2], was recently ratified by the Internet Engineering Task Force [3]. The iSCSI protocol is perceived as a low cost alternative to the FC protocol for remote storage [4][5][6][7]. It allows block level storage data to be transported over the popular TCP/IP network that can cover a wide area across cities and states. Therefore, the iSCSI lends itself naturally to a cost-effective candidate for remote mirroring making use of the available Internet infrastructure.

The viability of iSCSI protocol for remote mirroring depends, to a large extent, on whether acceptable performance can be obtained to replicate data to a remote site. While there are studies reported very recently on the iSCSI performance on LAN networks, campus networks, and emulated WAN [4][5][6][7], the open literature lacks technical data on the performance of the iSCSI protocol for remote mirroring over a realistic commercial WAN. The objective of this paper is two fold. First, we incorporate our new storage architecture to an iSCSI target to enhance write performance specifically for remote mirroring purposes. Secondly, we carry out measurement experiments to study the performance of the iSCSI protocol for remote mirroring on both a LAN and a commercial WAN network.

Our new storage architecture is referred to as DCD (disk caching disk) [8][9]. The idea is to use a log disk, called cache-disk, as an extension of a small NVRAM to cache file changes and to destage cached data to the data disk afterward when the system is idle.

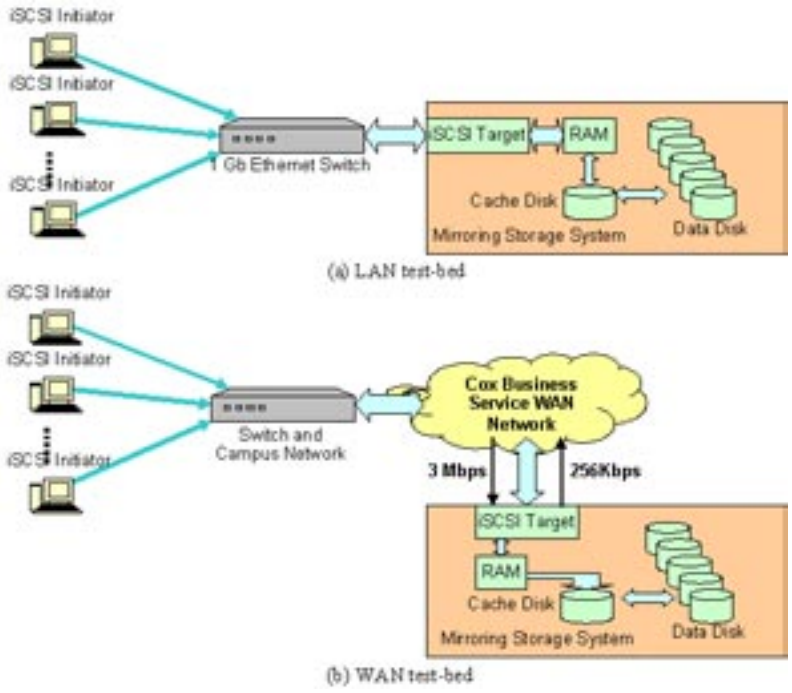


Figure 1. Experimental settings for performance measurements of iSCSI remote mirroring.

Small and random writes from the iSCSI network are first buffered in the small NVRAM buffer. Whenever the cache-disk is idle or the RAM is full, all data in the RAM buffer are written sequentially in one data transfer into the cache-disk. The RAM buffer is then made available quickly to absorb additional requests so that the two-level cache appears to the iSCSI network as a huge RAM with size of a disk. When the data disk is idle, a destage operation is performed, in which the data is transferred from the cache-disk to the normal data disk. Since the cache is a disk, it is cost-effective and highly reliable. In addition, the log disk is only a cache that is transparent to the file system and upper layer applications.

We incorporated our DCD into the iSCSI target program [10] that is used as a storage device for remote mirroring purpose. We carried out measurement experiments in two different settings: one inside our laboratory and the other over a commercial WAN through Cox-business Internet services. Our experiments with the real world commercial WAN give us great insightful experiences that may not be possible using emulated WAN in a laboratory [4][5][6][7]. We measured a variety of benchmarks and real world I/O traces widely used in the file system and storage system communities. Measured results show that the iSCSI protocol is a viable and cost-effective approach for remote mirroring. It is particularly useful to small to medium size organizations to deploy economical remote mirroring for tolerating site failures and disaster recovery when the cost of losing data matters.

2. Experimental Methodology

Figure 1 shows the two experimental setups for our experiments. Our first experiment was carried out inside our laboratory as shown in Figure 1(a). Several server hosts are connected to a mirroring storage system through an Intel NetStructure 470T Gigabit Ethernet switch. The server hosts act as iSCSI initiators while the mirroring storage system acts as an iSCSI target. Our second experimental setting is over a realistic commercial WAN through Cox Business Services. The iSCSI initiators in the LAN inside our laboratory are connected through our campus network and leased lines to the educational Internet. They are then connected to a website of a business office, Elake Data Systems, Inc., on the Cox Communications Inc.

cable network. The business office is used as a remote mirroring site that is located in a different town several miles away from our university campus. The down stream speed to the mirroring site is theoretically 3 Mbps and the upstream speed is theoretically 256 Kbps. Because of sharing of cables, the actual speed varies depending on network traffic and the time of a day. We found that during our experiments that the actual speeds vary from 40.7 KBps to 294 KBps. The cost of such a business connection is less than $100/month in New England area. We believe that such a connection and its cost represent a typical network connection for small to medium size businesses that mirror moderate amount of their business data at a remote site for failure tolerance and disaster recovery. As indicated in [1], the cost of leasing a WAN connection with speed of 155Mbps could cost about $460,000/year. Our objective here is to analyze the backup performance of the iSCSI protocol over an inexpensive WAN network where the iSCSI protocol is likely to be used for cost effectiveness.

All machines used in the experiments are Dell servers equipped with a single Pentium III 866MHz CPU, 512MB SDRAM, and the Intel Pro1000T Gigabit NIC (network interface card). We run Redhat 9 as the operating system with recompiled Linux kernel 2.4.20. Our iSCSI implementation is based on the implementation ref20_19b from University of New Hampshire [10]. The SCSI controllers we used are Adaptec AIC7899 Ultra 160 controller. All SCSI disks we used in our experiments are 18GB Seagate ST318452LW Ultra 160 disks. When two disks need to be connected to the same SCSI controller, we connect them to different channels.

At the iSCSI target, our DCD system is integrated with the iSCSI target software. Random and small write requests coming from the network are first buffered in the 32MB NVRAM buffer and the target acknowledges immediately to the server host for write completion. These small write requests are collected to form a log to be moved sequentially to the cache disk as soon as the cache disk is idle or the data in the NVRAM exceeds a predetermined watermark. As a result, there is always a room in the NVRAM buffer to accept new requests and the two-level hierarchy consisting of the RAM buffer and the cache disk appears to the network as a large RAM absorbing write requests quickly. In our current implementation, we use part of the host memory to emulate the NVRAM. Our previous experiments have shown that the maximum time before the data in the NVRAM buffer are moved to a disk is usually less than 100 milliseconds, which guarantees the safety of mirrored data even a DRAM buffer is used instead of the NVRAM provided that a UPS is used. Destaging operations between the cache disk and the data disk are done when the target storage system is idle.

Our remote mirroring software is based on the RAID1 code in the Linux kernel. There are two devices in a typical mirroring configuration, one is a primary device and the other is a secondary device. In a production implementation, there might be one or more spare devices available. Since we are interested in the performance of remote mirroring, we only consider the two-device configuration here. The mirroring software exports a block device to the operating system and applications that is very similar to the normal RAID1 device such as /dev/md0 or /dev/mdx. All read requests are sent to the primary device only while all write requests are sent to both devices. A write request sent from a upper layer is acknowledged as being finished only after the mirroring software receives acknowledgments from both devices. Therefore, our mirroring software falls into the category of "lock-step" or "synchronous" mode as defined by Ji et *al* [1]. The following is a list of hardware configurations that our experiments are based on:

- A primary SCSI disk with another SCSI disk as the mirroring device (S-S). We use a local SCSI disk to mirror another SCSI disk in the same system. This is a baseline configuration as a reference for comparison purpose.
- A primary SCSI disk with an iSCSI target device on a LAN as the mirror device (S-iL). In this configuration, we use an iSCSI target device on a LAN to mirror a local SCSI disk.
- A primary SCSI disk with an iSCSI target device on a WAN as the mirror device (S-iW). In this configuration, we use an iSCSI target device on a WAN to mirror a local SCSI disk.
- A primary SCSI disk with a DCD-enhanced iSCSI device as the mirroring device (S-iD). This

	TPC-C20	Financial-1	Financial-2
Number of requests	10,000,000	2,452,167	2,733,121
Number of write requests	2,965,750	1,502,641	480,529
Request size range	4096B-126,976B	1024B-17,116,160B	1024B-262,656B
Mean Request Size	47,063B	3,855B	2,508B
Write Size Range	4096B-126,976B	1024B-17,116,160B	1024B-262,656B
Mean Write Size	4,710B	4,838B	3,107B
Request per second	192	57	67
Write per second	57	35	12

Table 1. Characteristics of the traces used in our experiments

configuration incorporates our DCD technology into the iSCSI target device for mirroring purpose. Similarly, such DCD-enhanced iSCSI storage can be either located on the LAN designated as S-iDL, or located on the WAN referred to as S-iDW.

The workloads used in our experiments consist of popular storage benchmarks such as PostMark [11], IoMeter [12], and real world traces. PostMark [11] is a widely used [13][14] file system benchmark tool from Network Appliance, Inc.. It measures performance in terms of transaction rates in an ephemeral small-file environment by creating a large pool of continually changing files. Once the pool has been created, a specified number of transactions occur. Each transaction consists of a pair of smaller transactions, i.e. Create file or Delete file and Read file or Append file. Each transaction's type and files it affected are chosen randomly. The read and write block size can be tuned. On completion of each run, a report is generated showing some metrics such as elapsed time, transaction rate, total number of files created and so on.

The IoMeter is another highly flexible and configurable synthetic benchmark tool that is also widely used in various research works. IoMeter can be used to measure the performance of a mounted file system or a block device. For a mounted file system, it generates a large size file as the workplace and performs various configurable operations. For a block device, for example, a SCSI disk /dev/sda1, a RAID or mirroring device /dev/md0, IoMeter treats it as a normal file and directly reads or writes on it after opening it.

Besides the above benchmarks, real world traces are also used in our experiments. The first trace is TPC-C20 that is a block level trace downloaded from the Performance Evaluation laboratory at Brigham Young University. They had run TPC-C benchmark with 20 data warehouses using Postgres database on Redhat Linux 7.1 and collected the trace using their kernel level disk trace tool, DTB [15]. The other two traces are I/O traces from OLTP applications running at two large financial institutions, Financial-1 and Financial-2. They represent typical workloads of financial industries and are made available by Storage Performance Council in partnership with the University of Massachusetts who together are hosting a repository of I/O traces for use in the public domain [16]. Table 1 shows the characteristics of the traces that we use.

3. Results and Discussions

3.1. PostMark Results

Our first experiment is to measure the mirroring performances of PostMark benchmark. Figure 2 shows our measured times for finishing 100,000 transactions on 10,000 files of the PostMark benchmark. The total amount of data generated by the PostMark is about 700MB. We varied the proportion of write requests of all transactions in the benchmark between 50% and 30% and plotted them separately as shown in Figure 2. Let us first consider mirroring on the LAN network. As shown in Figure 2(a), it takes longer time to finish all the transactions when mirroring data using iSCSI target than mirroring data using a local

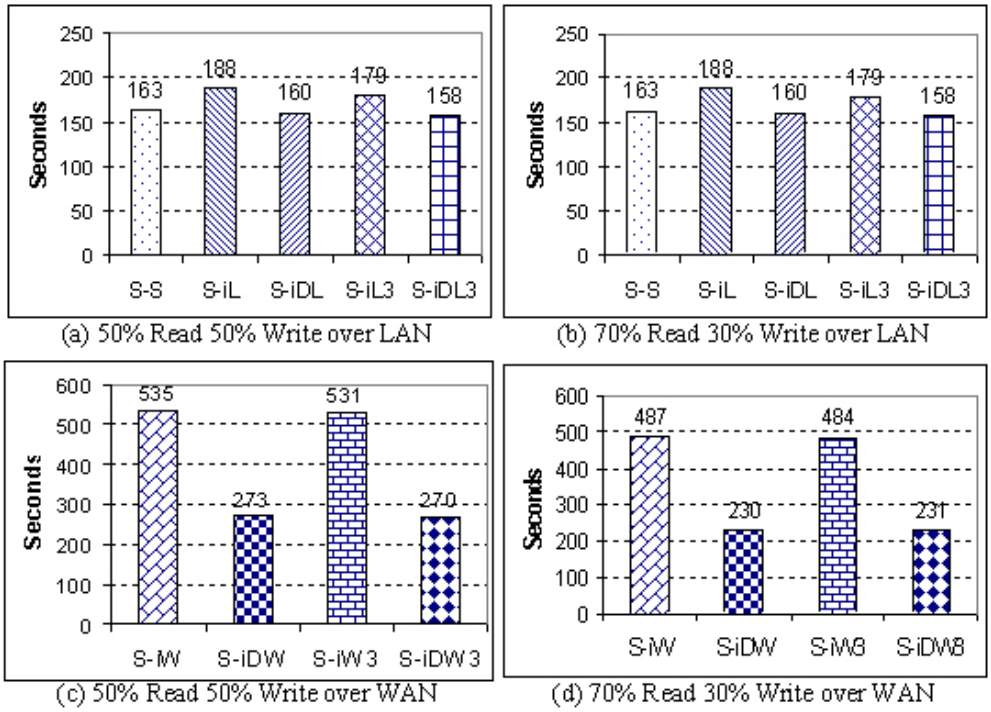


Figure 2. PostMark results: total time to finish 100,000 transactions on 10,000 files

Figure Legend for Figure-2 through Figure-4:
S-S: SCSI disk and SCSI disk mirroring.
S-iL: SCSI disk and iSCSI disk mirroring over a LAN.
S-iDL: SCSI disk and DCD-enhanced iSCSI disk mirroring over a LAN.
S-iL3: SCSI disk and iSCSI disk mirroring over a LAN with 3 parallel iSCSI connections.
S-iDL3: SCSI disk and DCD-enhanced iSCSI disk mirroring over a LAN with 3 parallel iSCSI connections.
S-iW: SCSI disk and iSCSI disk mirroring over a WAN.
S-iDW: SCSI disk and DCD-enhanced iSCSI disk mirroring over a WAN.
S-iW3: SCSI disk and iSCSI disk mirroring over a WAN with 3 parallel iSCSI connections.
S-iDW3: SCSI disk and DCD-enhanced iSCSI disk mirroring over a WAN with 3 parallel iSCSI connections.

SCSI disk. However, the time difference is not as significant as we initially expected, about 16% longer than that of local mirroring. It is interesting to observe that remote mirroring using the DCD-enhanced iSCSI disk takes shorter time than local disk mirroring. This is because DCD hides many seek times and rotation latencies of small writes by combining them into large logs to the cache disk, while with local disk mirroring, every write operation has to wait for two disk writes both requiring seek times and rotation latencies. Similar performance trends were observed for smaller write proportion as shown in Figure 2(b). While the total transaction times are shorter with 30% of writes because only write operations go to remote storage for mirroring, the relative difference between local mirroring and iSCSI mirroring keeps almost the same, about 15%.

iSCSI standard suggests that parallel iSCSI connections in a session may help improving its performance. To observe the effect of parallel connections on iSCSI performance, we measured transaction times with three concurrent connections as shown in the bars graphs marked with suffix 3. From our experiments, it seems that parallel connections do not show significant advantages over single connection. We believe that there could be two possible reasons that lead to the similar performances between parallel connections and single connection. One is the specific iSCSI implementation of UNH and the other is the low traffic intensity of PostMark. It remains open whether and how iSCSI can benefit from parallel and concurrent connections.

PostMark results for iSCSI mirroring over the WAN are shown in Figure 2(c) and Figure 2(d) for 50% writes and 30% writes, respectively. Clearly, synchronously mirroring data over the WAN increases the total transaction time dramatically. Compared to local mirroring, the total time to finish the same 100,000 transactions is more than tripled, from 189s to 535s. To gain some insight to why it takes such a long time to mirror over the WAN, we measured the RTT (round trip time) between our initiators and the target. While RTT fluctuates from time to time, we found the average RTT value to be around 14 ms. This round trip delay is on the same order as a disk operation including seek time, rotation latency and transfer time. For each write operation issued by the benchmark, it has to wait for the mirroring write that has to experience the RTT and a disk operation. As a result, the total transaction time is increased dramatically. The good news is that our DCD technology can help greatly. As shown in the figure, using DCD-enhanced iSCSI target for mirroring over the WAN, the total transaction time is reduced by half from that of pure iSCSI target. This improvement can be attributed to the effective caching of the DCD technology. Compared to the local disk mirroring, the DCD-enhanced iSCSI mirroring over the WAN shows about 40% increase in terms of total transaction time for pure synchronous/lock-step mode. Compared to iSCSI mirroring over a LAN, the difference is about 24%. We believe this is quite acceptable since this mirroring mode will essentially never lose data. Of course, one can allow some degree of asynchrony to obtain better performance. It would be interesting to compare our results here with the traditional FC-SAN mirroring, which we were not able to do because of lack of such FC-SAN facility. For smaller percentage of write operations as shown in Figure 2(d), similar relative performances were observed though the absolute transaction times are shorter because of smaller number of writes.

3.2. IoMeter Results

Our second experiment is to measure the mirroring performances of various configurations using IoMeter benchmark. We configured the IoMeter to generate two types of synthetic workloads, one is 100% writes and the other is 50% writes and 50% reads. Both workloads use random addresses with fixed block size of 4 KB. To minimize the truncation effect that will be explained shortly, we set duration of measurement for each point to 1 hour and reported the average write response time and the maximum response time for each configuration as shown in Figure 3.

Comparing the performance of the local mirroring with that of iSCSI mirroring on a LAN shown in Figure 3(a), IoMeter showed a much larger difference than PostMark did. We believe such a large difference in terms of average response time is the result of higher traffic intensity of the IoMeter benchmark. With 100% writes, the IoMeter continuously generates write requests one after another to both pri-

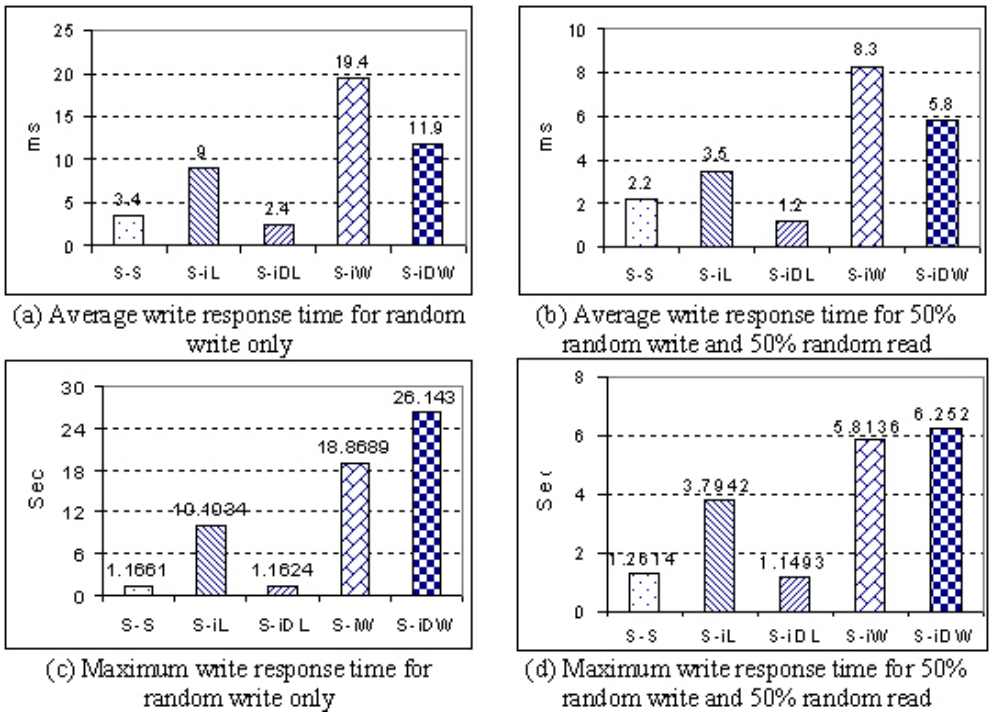


Figure 3. Testing results of IoMeter.

mary disk and the mirroring disk. As a result, it creates a lot of traffic over the network for mirroring data. One may argue that IoMeter generates requests back to back implying that it does not generate a new request until the previous request is acknowledged. However, because IoMeter uses asynchronous writes, it receives an acknowledgment as soon as the write is done in the file system cache. A queue of write requests may be formed giving rise to multiple write requests outstanding on the network. Such a queuing effect increases response time very rapidly. It is also this queuing effect that makes the performance improvement of the DCD-enhanced iSCSI mirroring more pronounced, almost 4 times improvement as shown in Figure 3(a). The queuing effect is substantially reduced if we decrease the write ratio from 100% to 50% as shown in Figure 3(b). In this case, we observed a smaller relative difference between local mirroring and iSCSI mirroring, and between iSCSI mirroring and DCD-enhanced iSCSI mirroring due to reduced write traffic.

For mirroring over the WAN network, the average write response time of using the iSCSI target is 19.4 ms and the response time of using the DCD-enhanced iSCSI target is about 11.9 ms as shown in Figure 3(a). The improvement of the DCD-enhanced iSCSI target over the iSCSI target is about 63%. Putting these results in a different perspective, a computer user would experience 19.4 milliseconds or 11.9 milliseconds delay on average if every write operation were synchronously backed up in a remote site using the iSCSI protocol. Note that these delays correspond to the workload that the user performs write operations continuously one after another. If 50% of continuous disk I/O operations were writes, the average response times would be lower, 8.3 ms and 5.8 ms for the iSCSI target and the DCD-enhanced iSCSI target, respectively as shown in Figure 3(b). Although the average response times are not outrageous, the maximum response times are noticeably large as shown in Figure 3(c) and (d). The maximum response time goes as high as 26 seconds for 100% writes and 6.3 seconds for 50% writes. These high maximum response times suggest that some kind of asynchronous mirroring should be desirable if write traffic is very high. In real world applications, the amount of write opera-

tions is limited and many organizations write less than 3 GB of data per year [17]. We noticed that while the DCD-enhanced iSCSI mirroring shows better average response time, its maximum response time is higher than other configurations. This is because that the target is very busy at that time point and it can hardly find idle time for destage operations. This result is consistent with our previous studies and observations that the DCD is beneficial only when the system can find idle time to carry out destage operations.

In our experiments with the IoMeter benchmark, we found several abnormal phenomena that are hard to explain. One of them is that the DCD-enhanced iSCSI mirroring shows significant lower average response times than the local SCSI disk mirroring. Another phenomenon is the truncation effect as mentioned in the beginning of this subsection. To understand these phenomena, we wrote a micro-benchmark program that continuously performs random writes of 4 KB blocks to a file with the system RAM being set to 256 MB. We measured the total times to finish 50,000 writes and 200,000 writes for three different cases: (1) synchronous writes, (2) asynchronous writes, and (3) asynchronous writes with forced flushing at the end. All the mirroring configurations are in a LAN environment. The asynchronous writes mimic the behavior of IoMeter because IoMeter records time stamp for each request individually and reports performance statistics of all finished transactions at the end of each test run. At the end of each test run, there may be writes done in the cache but no yet written into disks. The asynchronous writes with forced flushing at the end will ensure that all write requests generated in a test run are actually written into disks. As a result, the timing difference between the asynchronous writes and asynchronous writes with forced flushing is the truncation effect. Tables 2 and 3 show the measured results.

	S-S	S-iDL	S-iL
SYNC	273	273	649
ASYNC	161	153	208
ASYNC+Flush	171	155	515

Table 2. Total time in seconds to finish 50,000 transactions by the microbenchmark.

	S-S	S-iDL	S-iL
SYNC	1075	1075	2702
ASYNC	666	623	1666
ASYNC+Flush	677	671	2000

Table 3. Total time in seconds to finish 200,000 transactions by the microbenchmark.

For synchronous writes, we can see that both local mirroring and DCD-enhanced iSCSI mirroring have the same performance which is bounded by the slowest device that we believe is the primary disk. The iSCSI mirroring, however, takes longer time because the iSCSI target may be slower than the primary disk some times during the experiment. For asynchronous writes, the DCD-enhanced iSCSI mirroring shows advantages and even performs better than the local disk mirroring because the performance is no longer bounded by the primary disk because of cache effects. The performance difference increases as the number of transactions increases from 50,000 to 200,000. Similarly, the performance improvement of the DCD-enhanced iSCSI mirroring over iSCSI mirroring also increases as the number of transactions increases from 50,000 to 200,000.

The truncation effects are also shown in the tables as the performance differences between asynchronous and asynchronous with forced flushing. This difference can be as large as 148% as in the case of S-iL column for 50,000 requests. Such truncation effects did show when measuring the IoMeter performance with short measuring time of a few minutes. Therefore, we purposely enlarged the duration of each test run to 1 hour for each point of performance data. As shown in the tables, when we increase the length of test from 50,000 to 200,000 requests, the truncation effect reduced from 148% to 20% for the case of iSCSI mirroring. Similarly, the difference becomes negligible for the local mirroring case for longer test run. However, for some reason that we are not able to explain, the truncation effect increases a little bit (about 7%) for the DCD-enhanced mirroring after increasing the testing time. We believe that it may be the result of the destage process of the DCD system similar to the

phenomenon shown in Figure 3(c,d).

3.3. Traces Results

Figure 4(a) shows the average response times with the three types of the mirroring schemes for the three traces: Financial-1, Financial-2, and TPC-C. Similar performance results to that of the Postmark were observed. If the iSCSI protocol is to be used for remote mirroring, it is important to know the maximum delay caused by iSCSI protocol to determine which mirroring approach to take. We therefore plotted the minimum and maximum response times for three different mirroring schemes as shown in Figure 4(b) and 4(c). It is important to note that the real world workloads are quite different from the benchmarks in that one can always find idle time to take full advantage of the DCD technology. As shown in the figures, the maximum response time of the DCD-enhanced iSCSI is constantly the lowest among the three, implying its smooth and steady performance. The pure iSCSI, on the other hand, takes as much as 2 seconds maximally to mirror a write as shown in the figure. Therefore, it is advisable to use some kind of asynchronous mirroring approaches if an application cannot wait for such a long time.

The performance of remote mirroring over a realistic WAN network is shown in Figure 5. We drew here the response time plots of two traces, Financial-1 and Financial-2, mirroring over the WAN using the DCD-enhanced iSCSI only. As shown in this figure, the response times are noticeably much larger than those in the LAN. The maximum response time is as high as 13 seconds for Financial-1 and 5 seconds for Financial-2. The average response times are 733ms and 405ms for Financial-1 and Financial-2, respectively. However, majority of remote write operations can finish within one second for both cases as shown in this figure.

When converting the traces to SCSI requests, we used 16 parallel threads and each thread generates an SCSI request to the iSCSI initiator according to the time, address and size of one entry of the trace. After a request is issued, the thread waits for the response before generating another SCSI request. Because packet response times fluctuate greatly on a realistic WAN, it may happen that all our 16 threads are busy (blocked) waiting for responses while a new I/O request in the trace is supposed to be issued according to the trace timing. As a result, some write requests in the traces may get skipped until a thread is released. For the experiments reported in Figure 5, we observed that the initiators skipped about 6.55% of writes in the Financial-1 trace and 1.57% in the Financial-2 trace. Note that such skipping would not have happened in real applications but just slowed down the write process. It happened in our experiments because of our applying the traces collected in a fast environment to a slow environment.

To avoid skipping write requests in the traces, we carried out write coalescing at the iSCSI initiator side. Figure 6 shows the response time plots with write coalescing. The write coalescing size is 8 consecutive write requests. That is, each thread collects 8 consecutive writes and issues one write as a batch resulting from coalescing the 8 write operations. With this write coalescing, the 16 threads are able to issue 100% of write requests in both Financial-1 and Financial-2 traces. The response times plotted in Figure 6 correspond to the batch write operations. As shown in Figure 6, the response times for remote mirroring fluctuate but the maximum response times are lower than those in Figure 5. Majority of write mirroring can be done within a half of a second. Our measurement show that about 74.4% of mirroring can be done within one half of a second for financial-1 and about 90% of mirroring can be done within one half of a second for Financial-2. In order word, using the iSCSI protocol with our DCD architecture, one can mirror their business data on transaction-basis to a different town through a very inexpensive Internet connection (less than $100 per month). Mirrored data are safe within 6 seconds in the worst case and over 90% of data can be mirrored safely within half of a second if the transaction rate is as intensive as the Financial-2 trace. For TPC-C traces, we are still experiencing skipped requests of about 1.5% because of high traffic intensity. The response times for TPC-C shown in Figure 6(c) have about 98.5% of write operations of the entire trace.

During our WAN experiments, we also noticed that measured data varies from time to time because of different levels of network contentions. For example, results of daytime measurements may differ from night time and similarly weekdays from weekends.

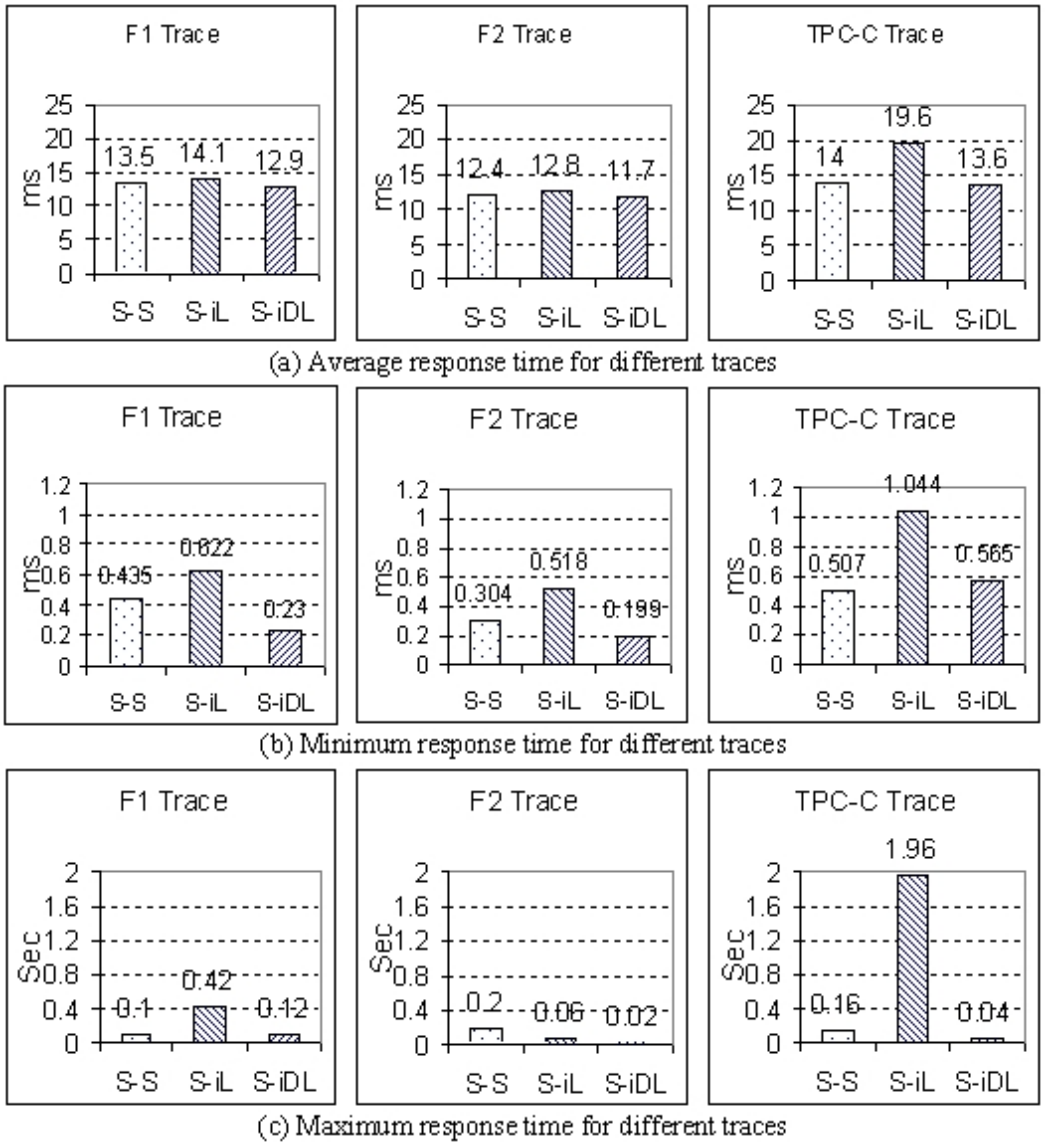


Figure 4. Response time for different mirroring schemes in LAN.

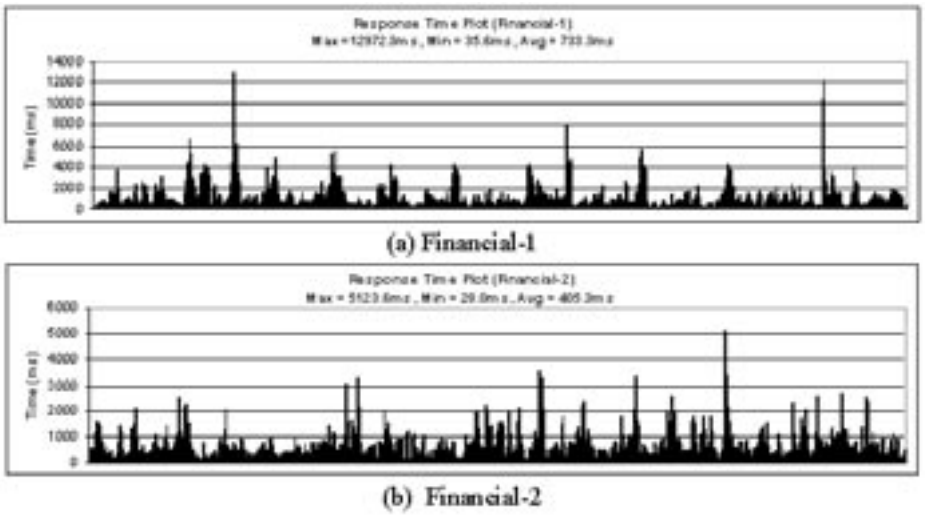


Figure 5. Response time plot of Financial-1 and Financial-2 traces over WAN.

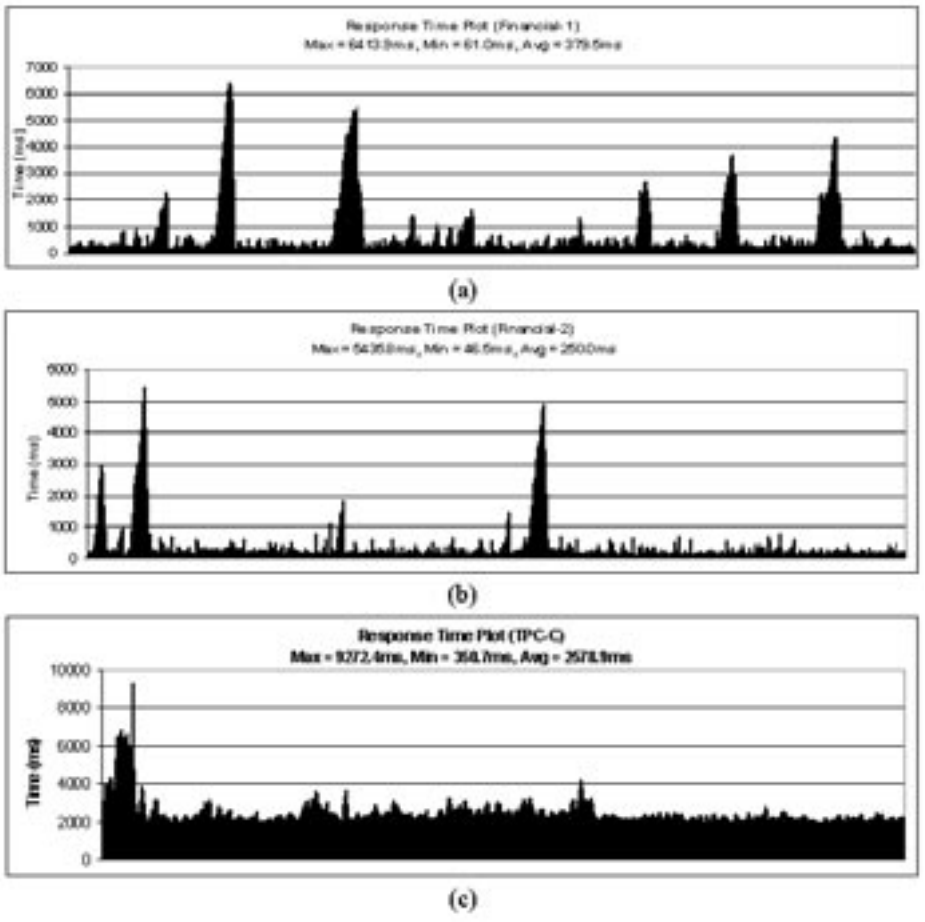


Figure 6. Response time with write coalescing (batch size=8 consecutive writes) over WAN.

4. Related Work

Remote mirroring is not new for data protection and disaster recovery [18]. Companies such as IBM, EMC, Veritas, Computer Associates, and Network Appliance Inc, etc., all provide their own proprietary solutions [19][20][21][22][23]. A good summary of various remote mirroring approaches can be found in [1] including a new asynchronous remote mirroring protocol called Seneca. Myriad [24] uses cross-site checksums instead of direct replication to achieve the same level of disaster tolerance as a typical single mirrored solution requiring less resources. Venti [25] uses a unique hash of a block's contents to act as the block identifier for read and write operations. Thus it enforces a write once policy and can act as a building block for constructing a variety of storage applications with backup and snapshot characteristics.

iSCSI [2] is an emerging IETF standard [3] to provide a mapping for the block level SCSI commands and data over existing TCP/IP networks. Such a technology is supposed to provide a cost-effective alternative to build low cost SAN systems. Meth and Satran [26] discussed some strategies they adopted when designing the iSCSI protocol. Many research works [5][6][7] concentrated on iSCSI performance evaluation in various hardware environments. Most of the WAN performance evaluations are carried out in emulated WAN environments instead of a realistic WAN. Nishan systems (now McData) and other vendors carried out a "Promontory Project" [27] to demonstrate the feasibility of iSCSI in long distance transmission with high speed WAN FC links. Tomonori and Masanori proposed optimization techniques in software iSCSI implementations [28]. A novel cache strategy was proposed to improve the iSCSI performance [4]. It was also proposed to use iSCSI for distributed RAID systems [29]. Many iSCSI software and hardware implementations and products are already available [10][30].

5. Summary and Conclusions

We have carried out measurement experiments to study the viability of using the iSCSI protocol for remote data mirroring for failure tolerance and disaster recovery. To enhance the performance of the iSCSI protocol, a new storage architecture called DCD is incorporated in the iSCSI target software. Measured results show that the DCD-enhanced iSCSI target storage provides smoother and better performance than local disk mirroring in a LAN environment. Experiments on a real commercial WAN show that response times fluctuate and can be very large. Still, data can be mirrored safely at a remote town within a second on average for typical online transaction processing such as Financial-1 and Financial-2 over an inexpensive Internet connection. Our experiments suggest that write coalescing on the initiator side can help in reducing network traffic. Our experience also indicates that measuring performance over a realistic WAN is quite different from an emulated WAN in a laboratory that is more controllable.

Acknowledgment

This research is supported in part by National Science Foundation under grants CCR-0073377 and CCR-0312613. Any opinion, findings and conclusions are those of authors and do not necessarily reflect the views of NSF. We would like to thank our shepherd, Ben Kobler, and the anonymous referees for their valuable comments. The authors would like to thank Elake Data Systems, Inc. (http://www.elakedata.com) for providing the remote site for our experiments.

References

[1] M. Ji, A. Veitch, and J. Wilkes, "Seneca: remote mirroring done write," in *Proceedings of the 2003 USENIX Annual Technical Conference*, San Antonio, TX, June 2003, pp. 253–268.

[2] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner, "iSCSI draft standard," http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt.

[3] C. Boulton, "iSCSI becomes official storage standard," http://www.internetnews.com/.

[4] X. He, Q. Yang, and M. Zhang, "Introducing SCSI-To-IP Cache for Storage Area Networks," in *Proceedings of the 2002 International*

Conference on Parallel Processing, Vancouver, Canada, Aug. 2002, pp. 203–210.

[5] W. T. Ng, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden, "Obtaining high performance for storage outsourcing," in *Proceedings of the Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002, pp. 145–158.

[6] S. Aiken, D. Grunwald, A. R. Pleszkun, and J. Willeke, "A performance analysis of the iSCSI protocol," in *IEEE Symposium on Mass Storage Systems*, San Diego, CA, Apr. 2003, pp. 123–134.

[7] Y. Lu and D. H. C. Du, "Performance study of iSCSI-based storage subsystems," *IEEE Communication Magazine*, vol. 41, no. 8, Aug. 2003.

[8] Y. Hu and Q. Yang, "DCD—disk caching disk: A new approach for boosting I/O performance," in *Proceedings of the 23rd International Symposium on Computer Architecture*, Philadelphia, Pennsylvania, May 1996, pp. 169–178.

[9] Q. Yang and Y. Hu, "System for destaging data during idle time," U.S. Patent 5 754 888, Sept. 24, 1997.

[10] UNH, "iSCSI reference implementation," http://www.iol.unh.edu/consortiums/iscsi/.

[11] J. Katcher, "PostMark: A new file system benchmark," Network Appliance, Tech. Rep. 3022, 1997.

[12] Intel, "IoMeter, performance analysis tool," http://www.iometer.org/.

[13] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, and E. Gabber, "Structure and performance of the direct access file system(DAFS)," in *Proceedings of USENIX 2002 Annual Technical Conference*, Monterey, CA, June 2002, pp. 1–14.

[14] J. L. Griffin, J. Schindler, S. W. Schlosser, J. S. Bucy, and G. R. Ganger, "Timing-accurate storage emulation," in *Proceedings of the Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002, pp. 75–88.

[15] Performance Evaluation Laboratory, Brigham Young University, "DTB: Linux Disk Trace Buffer," http://traces.byu.edu/new/Tools/.

[16] SPC, "Storage Performance Council I/O traces," http://www.storageperformance.org/.

[17] P. Desmond, "Going the distance for business continuity," http://www.nwfusion.com/supp/2003/ business/1020distance.html, Oct. 2003.

[18] C. Chao, R. English, D. Jacobson, A. Stepanov, and J. Wilkes, "Mime: a high performance parallel storage device with strong recovery guarantees," Hewlett-Packard Laboratories, Tech. Rep. HPL-CSP-92-9 rev1, Nov. 1992.

[19] IBM, "DFSMS SDM Copy Services," http://www.storage.ibm.com/software/sms/sdm/.

[20] EMC, "Symmetrix remote data facility (SRDF)," http://www.emc.com/.

[21] Veritas, "VERITAS storage replicator," http://www.veritas.com.

[22] Computer Associates, "BrightStor ARCserve backup," http://www.ca.com.

[23] Network Appliance Inc., "SnapMirror software: Global data availability and disaster recovery," http://www.netapp.com/.

[24] F. Chang, M. Ji, S.-T. A. Leung, J. MacCormick, S. E. Perl, and L. Zhang, "Myriad: Cost-effective disaster tolerance," in *Proceedings of the Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002.

[25] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," in *Proceedings of the Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002.

[26] K. Z. Meth and J. Satran, "Design of the iSCSI protocol," in *IEEE Symposium on Mass Storage Systems*, San Diego, CA, Apr. 2003.

[27] McData, "The Promontory project: Transcontinental IP storage demonstration," http://www.mcdata.com/splash/nishan/.

[28] F. Tomonori and O. Masanori, "Performance optimized software implementation of iSCSI," in *International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, New Orleans, LA, Sept. 2003.

[29] X. He, P. Beedanagari, and D. Zhou, "Performance evaluation of distributed iSCSI RAID," in *International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, New Orleans, LA, Sept. 2003.

[30] Microsoft, "Microsoft delivers iSCSI support for Windows," http://www.microsoft.com.

Simulation Study of iSCSI-based Storage System*

Yingping Lu, Farrukh Noman, David H.C. Du
Department of Computer Science & Engineering,
University of Minnesota
Minneapolis, MN 55455
Tel: +1-612-625-4002, Fax: +1-612-625-0572
Email: {lu, noman, du }@cs.umn.edu

Abstract

iSCSI is becoming an important protocol to enable remote storage access through the ubiquitous TCP/IP network. Due to the significant shift in its transport mechanism, iSCSI-based storage system may possess different performance characteristics from the traditional storage system. Simulation offers a flexible way to study the iSCSI-based storage system. In this paper, we present a simulation work of iSCSI-based storage system based on ns2. The storage system consists of an iSCSI gateway with multiple targets, a number of storage devices that are connected to the storage router through FC-AL and clients, which access the target through iSCSI protocol. We present the system model, the implementation of the components, the validation of the model and performance evaluation based on the model. Coupled with the rich TCP/IP support from ns2, the simulation components can be effortlessly used for the performance and alternatives study of iSCSI-based storage systems, applications in broad configurations.

1. Introduction

The iSCSI protocol [1][2] has emerged as a transport for carrying SCSI block-level access protocol over the ubiquitous TCP protocol. It enables a client's block-level access to remote storage data over an existing IP infrastructure. This can potentially reduce the cost of storage system greatly, and facilitate the remote backup, mirroring applications, etc. Due to the ubiquity and maturity of TCP/IP networks, iSCSI has gained a lot of momentum since its inception.

On the other hand, the iSCSI-based storage is quite different from a traditional one. A traditional storage system is often physically restricted to a limited environment, e.g. in a data center. It also adopts a transport protocol specially tailored to this environment, e.g. parallel SCSI bus, Fibre Channel, etc. These characteristics make the storage system tend to be more robust, and achieve more predictable performance. It is much easier to estimate the performance and potential bottleneck by observing the workload. While in an iSCSI storage, the transport is no longer restricted to a small area. The initiator and the target can be far apart. The networking technology in between can be diverse and heterogeneous, e.g. ATM, Optical DWDM, Ethernet, Wireless, satellite, etc. The network condition can be congested and dynamically changing. Packets may experience long delay or even loss and retransmission, etc. Thus, the situation facing the iSCSI storage is quite different from the traditional one.

To take advantage of iSCSI protocol to build iSCSI storage systems, we need to better understand the iSCSI characteristics, e.g. the performance characteristics in various

* This work was supported by DISC from DTC of UoM, and gifts from Intel and Cisco.

networking situations, the impact of network transmission error or network component fault to the storage access performance and robustness, the relationship between iSCSI and underlying TCP/IP protocol, etc.

The common way to study the performance characteristics about the iSCSI storage system is through the real performance measurement. Real measurement can be pretty accurate. Papers [4][5][6] represent this endeavor. However, the measurement approach is often restricted to the physical equipments and settings. In a lot of times, the software or hardware of the equipment is not open. Thus a tester cannot adjust parameters, or try an alternative algorithm, etc. in the performance study. In this regard, a simulation approach offers much more flexibility. Once the simulation components have been implemented, it is very easy to construct test configuration, configure parameters of interest, or add an alternative algorithm, etc. to study the iSCSI related issues.

To our best knowledge, no simulation model has been built for iSCSI protocol. Our goal of this work is to establish a simulation model for iSCSI-based storage system for the study of the characteristics of iSCSI-based storage system. In addition, we also study the interactions between the iSCSI and TCP layer to better support the iSCSI access.

We use network simulator NS2 [9] to implement the iSCSI simulation model. NS2 is an event driven simulator widely used in the research of networking arena. It provides substantial support for the simulation of TCP/UDP, routing, and multicast protocols over wired and wireless networks. To validate the simulated model, we also conduct the real performance measurement and compared the simulation results with the real performance data. In addition, we also examine the different TCP parameters and investigate how they affect the iSCSI performance based on the simulation model.

This paper is organized as follows: Section 2 presents the simulation model for the iSCSI-based storage system. Section 3 describes the iSCSI implementation in NS2. Section 4 presents empirical validation of the model. In Section 5, we analyze the performance results of iSCSI model. Section 6 reviews the related work. Finally we conclude this paper in Section 7.

2. Simulation Model

2.1. A Typical iSCSI Storage System Model

Figure 1 shows a typical iSCSI-based storage system model used in the simulation. In this model, an initiator generates SCSI requests, which is encapsulated into iSCSI messages (protocol data unit or PDU). These PDUs are then transmitted over TCP/IP network and routed to an iSCSI storage gateway.

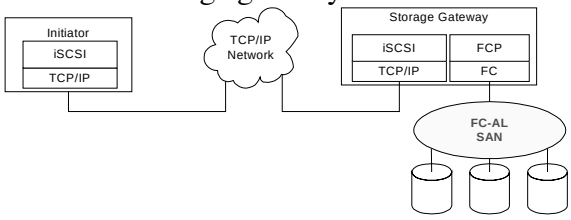


Figure 1 The storage system model

The storage gateway has both TCP/IP networking interface and FC-AL interface. TCP/IP interface provides iSCSI connection for an initiator to access through IP network, while FC-AL interface is used to connect to a SAN storage subsystem. In this SAN

environment, the gateway serves as an initiator to the FC-enabled disks. It uses the SCSI over FC encapsulation protocol (FCP) to access the disk devices through Fibre Channel.

2.2. The iSCSI Data Transfer Model

Figure 2 shows the iSCSI architecture model. iSCSI builds on top of TCP transport layer. For an iSCSI initiator to communicate with a target, they need to establish a session between them. Within a session, one or multiple TCP connections are established. The data and commands exchange occurs within the context of the session.

Figure 3 shows iSCSI command execution by illustrating a typical Write command. The execution consists of three phases: Command, Data and Status response. In the Command phase, The SCSI command (in the form of Command Descriptor Block (CDB)) is incorporated in an iSCSI command PDU. The CDB describes the operation and associated parameters, e.g. the logical block address (LBA) and the length of the requested data. The length of the data is bounded by a negotiable parameter "MaxBurstLength". During the Data phase, the data PDUs are transmitted from an initiator to a target. Normally, the initiator needs to wait for "Ready to Receive (R2T)" message before it can send out data (solicited data). However, both initiator and target can negotiate a parameter "FirstBurstLength" to speed up the data transmission without waiting. FirstBurstLength is used to govern how much data (unsolicited data) can be sent to the target without receiving "Ready to Receive (R2T)". A R2T PDU specifies the offset and length of the expected data. To further speed up the data transfer, one data PDU can be embedded in the command PDU if "ImmediateData" parameter is enabled during the parameter negotiation. This should be very beneficial for small write operation. The Status PDU is returned once the command is finished.

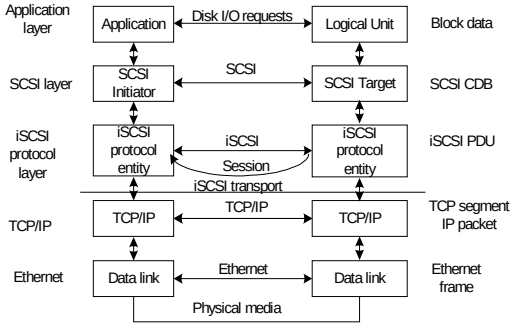


Figure 2. The iSCSI model

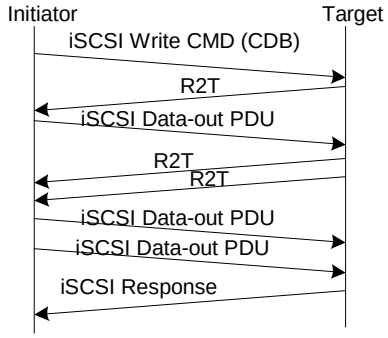


Figure 3. The command execution sequence

Finally, these messages are encapsulated into TCP/IP packets, where the packet size is bounded by MSS (maximum segment size) in TCP. MSS is determined by the smallest frame size along the path to the destination. Within an Ethernet LAN, the maximum frame size is 1500 bytes (Gigabit Ethernet supports Jumbo frame). The MSS is 1460 bytes (40 bytes for IP and TCP header). When the iSCSI PDU size is greater than the segment size, the PDU is further fragmented into smaller packets.

The size of iSCSI parameters like: MaxBurstLength, FirstBurstLength and PDU size all have certain impact to the iSCSI performance. However, the iSCSI performance is also significantly affected by the underlying TCP flow control, congestion control mechanism. We will also examine the effect of these parameters.

2.3. Disk Model

We use *Seagate ST39102FC Cheetah 9LP* disk as the storage device. The disk access time $T_d = T_{ds}+T_{dr}+T_{dt}$, where T_{ds} is the disk seek time, which is determined by the difference of current cylinder and the target cylinder; T_{dr} is the disk rotation latency, which is determined by the difference of the current sector when the disk head moves to the target cylinder and the first sector of the intended access. Disk transfer time T_{dt} is determined by the number of data blocks transferred. When the data size is large, the requested data may span more than 1 track (cross disk surface) or even 1 cylinder. In that case, we also add the head switch or cylinder switch time into the access time. We consider the disk has enough buffers to hold the requested data.

The disk not only handles the block data access, it also handles the data transmission. The disk has built-in FC-AL logic and interface. As a normal FC node, it has a physical address and needs to participate the arbitration phase to win the channel before transferring data between the gateway and the disk.

2.4. FC-AL Interconnect

Fibre Channel is a popular networking protocol to construct storage area network. It supports switching fabric, loop and point-point construct. FC-AL aims at loop topology where up to 126 FC-AL nodes (hosts and disk devices) are connected to a shared loop. A node obtains the access to the loop through arbitration. The arbitration is determined by the physical addresses of participating nodes. When a node wins arbitration, it opens a connection to its destination node. A node closes a connection and releases the control over a loop when its transfer has finished. In our environment, the disk devices and storage gateways are FC-AL nodes.

On top of the Fibre Channel transport, similar to iSCSI, FCP protocol maps SCSI protocol onto the Fibre Channel protocol. Each SCSI protocol is also performed through three phases, FCP-CMD frame for command transfer, FC-Data frame for data transfer, and FCP-response frame for status transfer. Since the maximum frame size is 2048bytes, a SCSI command may require multiple FCP-Data frames to transfer all requested data. For the SCSI Write operation, FCP-XFER-Ready frame is also used for the flow control between the initiator and disk device.

The speed of FC-AL can be 1Gb/s, up to 2 Gb/s. Since it adopts 8B/10B encoding, the actual bandwidth is 100MB/s (Mega bytes per second) and 200MB/s respectively. In our simulation, we assume the speed to be 1Gb/s. We use a central module FC to handle the channel arbitration. All participating nodes (disk devices, the gateway) are required to register to this module. When a node requires channel, it submits a request to FC. FC module determines who wins the arbitration.

2.5. Storage Gateway Model

The storage gateway works as a bridge between two protocols: iSCSI and FCP. It hosts the targets of the iSCSI and the initiators of the FC storage. In the meantime, it manages the targets, their access control and their related sessions. It also administers the mapping between a target and its constituent disk devices.

In the simulation, the disk devices should be “attached to” (add an pointer in) a target. Each target maintains two interfaces: iSCSI on top of TCP agent and the FCP on top of Fibre Channel interface. An outstanding command queue for the each session glues these

two interfaces together. Each command item in the queue contains the CDB and other status information. When a new command arrives from iSCSI interface, it is placed into the command queue. The command is further sent to the disk device through the FCP interface when the number of outstanding commands falls below the threshold in the target disk. When a command completes, it receives an FCP-RSP frame from the disk. Upon receiving this frame, the target sends out an iSCSI Response PDU to the actual initiator, in the mean time, the command is removed from the queue. However, the commands within a session are completed in order.

3. Simulating iSCSI in NS2

3.1. iSCSI Nodes

In our simulation there are three types of nodes: Initiator, Target and gateway node. Figure 4 shows these nodes and their related components. A target node is the peer of an initiator node. The gateway node hosts one or multiple target nodes.

iSCSIInitiator and iSCSITarget are the node applications running on the initiator and target respectively. Within these nodes, iSCSIInitiatorSession and iSCSITargetSession, derived from iSCSISession, perform the session tasks. Similarly, iSCSIInitiatorConnection and iSCSITargetConnection, derived from iSCSIConnection, perform connection tasks. A iSCSISession can open multiple iSCSIConnections.

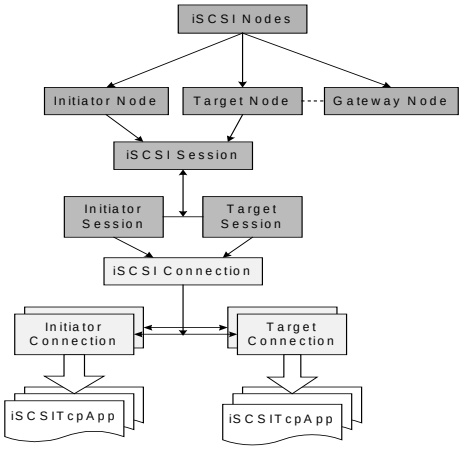


Figure 4. Hierarchy of iSCSI Node

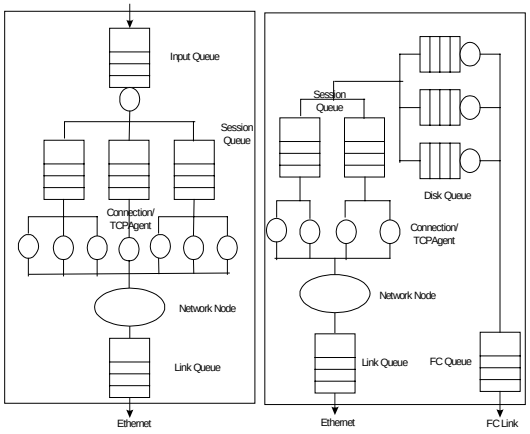


Figure 5 Queuing models (a) Initiator (b) Target

Each iSCSIConnection has its own iSCSITcpApp object through which data is sent and received. We use FullTCP agent for the iSCSIConnection application.

3.2. Queuing Models

Figure 5 shows the queuing models in the simulation. Fig. 5(a) is the model for an initiator. It has an input queue to receive workload (SCSI requests). Under the input queue are several Session queues. Each session possesses a queue for the outstanding SCSI commands. The maximum number of outstanding commands is configurable. The commands in each session are processed by their corresponding connections and then passed to their TCP agents. The link queue at the bottom is the network node's link queue.

Fig. 5(b) shows the Target's queuing model. Four types of queues exist: session queue, link queue, disk queue and FC queue. Similar to the Initiator node, link queue is for the Ethernet link. Per-session queue is for outstanding commands. Each disk has a disk queue, which holds outstanding commands for each disk in the target. Disk queue makes the interaction between the target and disk simpler. FC queue holds the requests to access the FC link. Moreover, each disk itself also has a command queue. The number of outstanding commands is configurable.

3.3. Implementation

Figure 6 shows a typical setting of an iSCSI system implemented in NS2. The system comprises three parts: the initiator, which generates workload; the TCP/IP network, which can be easily configured based on existing NS2 components; and the gateway in conjunction with target disk devices.

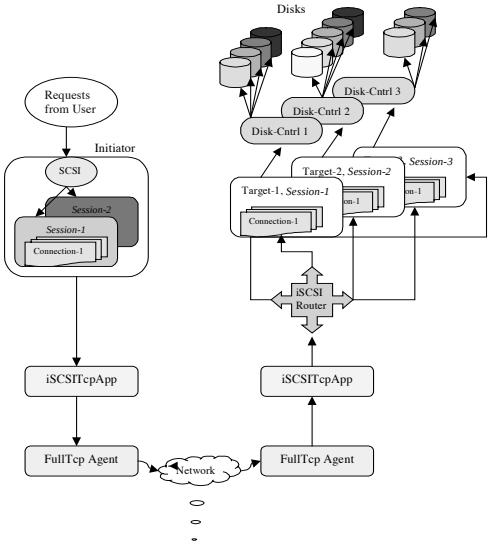


Figure 6. iSCSI system based on NS2

All these components are implemented in C++ to achieve high efficiency. The Otcl script in NS2 is used to setup the simulation environment. It creates network nodes and network topology, creates initiator and gateway nodes, creates targets and disks and attaches disks to target. These C++ components also expose a number of configurable parameters such as iSCSI parameters, disk parameters, to the Otcl. This makes the change of test setting very convenient.

After the test setting is constructed, the script then invokes login method in the initiator to connect to the gateway. The initiator enters Login phase. Eventually it acquires the targets and LUNs in the target from the gateway. The initiator finally creates a session with each target. The number of sessions and the number of connections in a session are also configurable parameters. The SCSI Read or Write requests (workload) are then passed to the initiator to carry out.

The workload for the each test is generated in a separate Otcl script. We have developed a workload generator program that generates requests of even distribution and Poisson distribution. The disk id is also evenly distributed among the specified range of disks. In addition, we also apply the trace file to the initiator to see how the actual application data affect the performance.

4. Empirical Validation

To verify the simulation model, we compare it with iSCSI performance measurement data obtained from a real iSCSI setting. In this real setup, Initiator communicates with a Cisco SN5420 iSCSI gateway through campus network. The network connection is the 100Mb/s Ethernet. There are 4 Seagate 39102FC disks are accessed. The round trip time in terms of network distance is approximately 2ms.

The test involves reading and writing a burst data of sizes from 1K, 4K, 16K to 64K bytes under light load and heavy load with a PDU size of 8KB.

1) A comparison of the real iSCSI access latency and NS modeled iSCSI access latency is shown in Figure 7. Both the light load (Each time only one thread is sending data request) and heavy load (4 threads is sending requests) are tested. The delay patterns for both figures shown above are approximately similar and follow the similar rate of increase. With small data burst size, the difference is within 2%, for the data burst size of 64K, the difference is within 6%.

2) In another test, the burst sizes vary from 4K to 64K under heavy load conditions. Figure 8 shows close approximation in throughput. The NS model provided a little higher throughput because of ideal environment conditions.

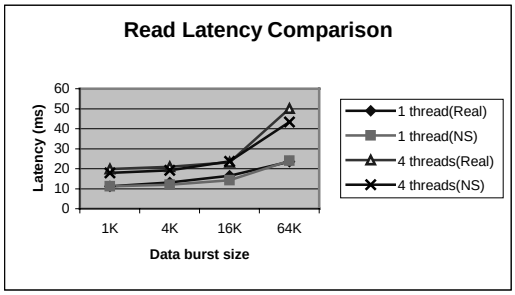


Fig. 7 Response time comparison

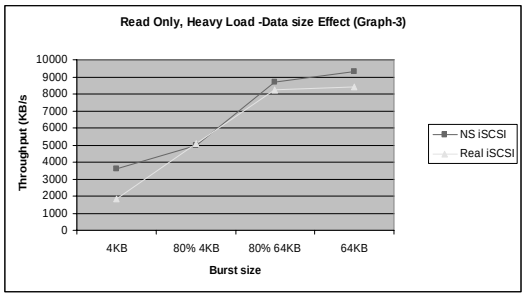


Fig. 8 Throughput comparison for heavy load

The results from the test provide reasonable assurance that the NS model can closely approximate a real iSCSI installation. To support the validity of model more thorough analysis can be done with different test scenarios.

5. Performance Analysis

In this section, we present the results of the effect of iSCSI parameters in iSCSI layer and TCP parameters in TCP layer to iSCSI data access performance.

5.1. The Effect of the iSCSI Parameters

We first examine the effect of different iSCSI PDU sizes. Figure 9 shows the read throughput with varying PDU sizes. The parameters involved in this simulation include data PDU sizes from 0.5KB to 8KB and max burst sizes from 1KB to 4MB. It is found out that at larger burst data size, the PDU size makes difference. For a large burst size, e.g. 1MB, with the PDU size of 8K, there will be 128 PDUs, while with PDU size of 1K, then there will be 1024 PDUs. More PDUs cause more R2T messages, and potentially more waiting for the R2T signals. From the figure, we observe the better performance for larger PDU size as data size increases.

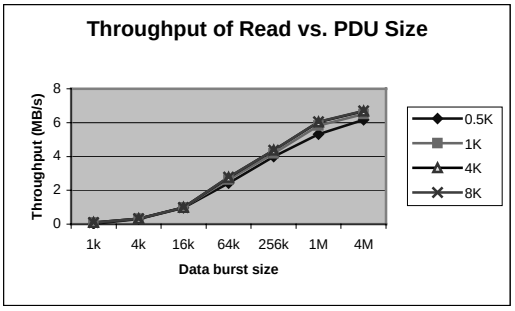


Fig. 9 The effect of iSCSI PDU size

5.2. The Effect of Network Parameters

In this subsection, we investigate how the network parameters like TCP window size, MSS (Maximum Segment Size) and link delays affect the iSCSI performance.

We first examine the effect of MSS size. In the test setting, the TCP window assumes the default value of 20. The link is a Gigabit Ethernet with delay from 10us to 50ms. The MSS sizes are 296B, 576B and 1500 bytes respectively. Figure 10 shows the achieved throughput with varying MSS sizes.

Normally, for a given delay and MSS size, the maximum throughput that can be achieved is approximately one window per round trip time, i.e. (MSS * window)/2*delay, which implies that throughput is inversely proportional to the link delay for the given MSS.

This figure shows that the throughput decreases quickly for smaller MSS sizes, whereas higher MSSs show a gradual decrease even for higher link delays. For link delays less than 1ms, the MSS size does not have much effect on the throughput this is because at short network link delay, bandwidth-delay product is small. The acknowledgement comes back very fast. As the link latency continues to increase, the throughput drops gradually, thus the link utilization is also getting lower. We need more parallelism to take advantage of the link bandwidth. The use of multiple connections may help. Adding more disks will increase the disk I/O bandwidth. The RAID system may also increase the disk access performance.

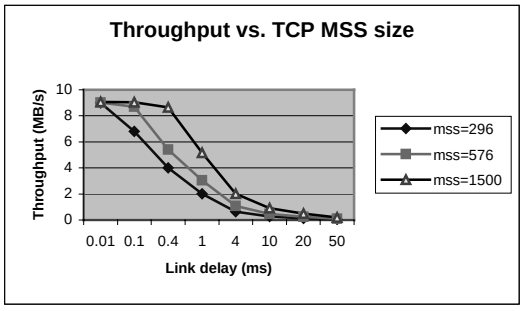


Fig. 10 Throughput vs. MSS size

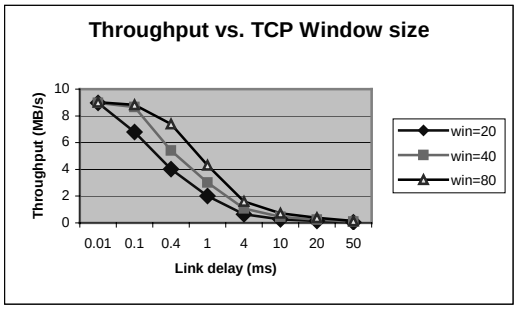


Fig. 11 Throughput vs. Window size

We then examine how the TCP window variation affects the throughput. Three different TCP window sizes (20, 40 and 80) are used. The MSS size is fixed at 296B as shown in the Figure 11. The result shows that the throughput increases with the increase of TCP window. At the short network link latency (e.g. LAN environment) of 0.4ms, the throughput is about 4MB/s for window size 20, but when the window size increases to

80, the throughput reaches to 6.8MB/s. However, with the increase of network latency (long fat network), the window size is too small, and the throughput reduces significantly.

6. Related Work

There have been several simulation work related to disk and storage subsystem. Paper [7] gave an introduction about the disk drive modeling and simulation. It described the principle of a disk drive and present a formula to compute disk seek time. Project DiskSim [8] provides an open source code, which can extract the disk parameters of different disk drives. We benefit a lot from their work in the disk simulation. Paper [10] modeled a disk controller and studied some more advanced features like caching. In paper [11], a Storage Area Network (SAN) is simulated. In this SAN, Myrinet is used to connect the storage devices and servers (initiators).

On the other hand, iSCSI protocol represents a different SAN paradigm, i.e. it uses the ubiquitous TCP protocol as the SCSI command and data transport. There are several studies in iSCSI performance and characteristics. Papers [4][5][6] presented the iSCSI performance measurement and evaluation under different scenarios. However, due to the diversity of network configuration and the impact of the underlying TCP network, it is crucial to build up the simulation model for the iSCSI environment for the iSCSI-related research. Our work incorporates the rich feature of ns2 in supporting the TCP/IP networking, and implements all related components including disk model, iSCSI protocol, FC-AL protocol and iSCSI target. This can be used to easily construct a flexible iSCSI-based storage system to facilitate the iSCSI related research.

7. Conclusion

We have presented a simulation model for the iSCSI-based storage system. The model includes all components for constructing an iSCSI-based storage system. In order to meet the requirements of extensibility and flexibility, we make the components modular and generic. The whole system is composed into several components. These components can be easily replaced or extended.

In order to validate the implementation, we also conducted real measurement and compared the simulation results with the real measurement results under the same settings. It turns out that the performance results in our model are close to that of real measurement.

With the simulation model, we further conducted the performance characteristics study to examine how the iSCSI parameters and the underlying TCP parameters affect the iSCSI performance. Our results show that with larger burst data size, the larger PDU size will help the throughput. Increasing TCP window size and MSS also affects the end-to-end performance. But this effect is more pronounced for higher link delays.

In the future, we'll further study the iSCSI storage system in a diverse network such as fat network (long latency, high bandwidth), wireless network, we'll also examine the impact of the underlying TCP protocol on the upper-level SCSI access in terms of performance and resilience based on the simulation model.

Acknowledgement

The authors thank to Avinashreddy Bathula who helped the initial work of this project. We are also grateful for the help offered by our shepherd Randal Burns.

References

[1] Julian Satran, et al. iSCSI Specification, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt, Jan. 2003

[2] Kalman Z. Meth, Julian Satran, Design of the iSCSI Protocol, IEEE/NASA MSST 2003, Apr. 2003.

[3] K. Voruganti; P. Sarkar, An Analysis of Three Gigabit Networking Protocols for Storage Area Networks, 20th IEEE International Performance, Computing, and Communications Conference", April 2001

[4] S. Aiken, D. Grunwald, A. Pleszkun, Performance Analysis of iSCSI protocol, IEEE/NASA MSST 2003, Apr. 2003.

[5] Y. Lu, D. Du, Performance Evaluation of iSCSI-based Storage Subsystem, IEEE Communication Magazine, Aug. 2003

[6] S. Tang, Y. Lu, D. H.C. Du: Performance Study of Software-Based iSCSI Security. IEEE Security in Storage Workshop 2002: 70-79

[7] C. Ruemmler and J. Wilkes. An Introduction to Disk Drive Modeling. IEEE Computer, Vol. 27, No. 3, 1993, pp 17-28

[8] G. Ganger, B. Worthington, Y. Patt, The DiskSim Simulation Environment, http://www.pdl.cmu.edu/diskSim/index.html

[9] The network simulator ns2, http://www.isi.edu/nsnam/ns/

[10] M. Uysal, G. A. Alvarez and A. Mechant, A Modular, Analytical Throughput Model for Modern Disk Arrays, MASCOTS-2001, Aug. 2001

[11] X. Molero, F. Silla, V. Santonja, Jose Duato, Modeling and Simulation of Storage Area Networks, MASCOTS-2000, Sep. 2000, pp. 307

[12] D. Anderson, J. Dykes and E. Riedel, More Than An Interface – SCSI vs. ATA, Proc. of the 2nd Annual Conference on file and Storage (FAST), Mar. 2003

Comparative Performance Evaluation of iSCSI Protocol over Metropolitan, Local, and Wide Area Networks

Ismail Dalgic Kadir Ozdemir Rajkumar Velpuri Jason Weber
Intransa, Inc
2870 Zanker Rd.
San Jose, CA 95134-2114
Tel: +1-408-678-8600
{ismail.dalgic, kadir.ozdemir, rajkumar.velpuri, jason.weber}@intransa.com

Helen Chen
Sandia National Laboratories, California
PO Box 969
Livermore, CA 94551
Tel: +1- 925-294-3000
hycsw@ca.sandia.gov

Umesh Kukreja
Atrica, Inc.
3255-3 Scott Blvd
Santa Clara, CA 95054
Tel: +1-408-562-9400
umesh_kukreja@atrica.com

Abstract
We identify the tunable parameters of iSCSI and TCP that affect the performance characteristics for local, metropolitan, and wide area networks. Through measurements, we determine the effect of these parameters on the throughput. We conclude that with the appropriate tuning of those parameters, iSCSI and TCP protocols maintain a good level of throughput for all types of networks.

1. Introduction

iSCSI [1] is a promising new technology, which overcomes the distance limitations of other storage networking technologies such as Fibre Channel and Infiniband, and thereby enables globally distributed mass storage systems. Wide Area Ethernet services, at the same time are emerging as a strong contender for wide area connectivity among multiple enterprise locations. While some studies exist on the performance characteristics of the iSCSI protocol [2] [3], the performance characteristics for metropolitan area and wide area networks are yet to be understood. The iSCSI protocol and the underlying TCP/IP and Ethernet protocols have some configurable parameters which impact performance. In this paper, we investigate the effect of some of these parameters on iSCSI throughput.

2. Parameters

At the iSCSI level, the parameters of interest from a performance point of view are: (i) command request size, (ii) iSCSI command window credit amount, (iii) the number of simultaneous iSCSI connections in a session, and (iv) the

option of sending solicited vs. unsolicited data. The command request size, the command window credit amount, and the number of simultaneous connections may impact both read and write performance. The choice of solicited vs. unsolicited data may impact write performance, but it has no impact on reads.

At the TCP/IP level, the most important parameters are the send and receive window sizes especially in networks with a large bandwidth-delay product such as high speed WANs.

The iSCSI command request size is the amount of data that is sent or received as part of a SCSI command encapsulated in iSCSI. The iSCSI command window credit amount, dynamically set by the target, determines the maximum number of iSCSI commands that can be outstanding at a given time. The product of these two parameters will determine the maximum amount of data that can be pipelined in the network to deal with the network latency. Increasing this amount will generally improve throughput.

The primary reason for iSCSI to support multiple connections per session is to take advantage of trunking in Gigabit LAN switches [4]; each TCP connection may utilize a different link, thus improving the overall throughput of the session. However, even on a WAN or MAN link where only a single path is available between an initiator and a target, the number of simultaneous connections in an iSCSI session may impact the performance due to the behavior of the TCP protocol where each TCP connection adjusts its transfer rate so as to share fairly a congested path. By allowing multiple connections per iSCSI session, the iSCSI traffic is effectively given priority over other TCP traffic. Furthermore, a packet loss in a TCP connection triggers the TCP slow-start and congestion avoidance algorithms, resulting in a drop in the throughput which takes some time to reach back to the maximum possible level [5]. By using multiple connections in a session, the overall impact of this temporary drop in throughput is reduced. On the other hand, the iSCSI protocol has to obey the SCSI command ordering rules that may reduce the parallelism among multiple connections.

As far as solicited vs. unsolicited data transfer is concerned, three independent parameters determine the transfer type: *FirstBurstLength*, *MaxBurstLength,* and *MaxRecvDataSegmentLength* [1]. *FirstBurstLength* determines the maximum amount of unsolicited data that the initiator can send per command. *MaxBurstLength* determines the maximum amount of solicited data that the initiator can send per command. *MaxRecvDataSegmentLength* is the maximum data segment size that can be sent in each protocol data unit (PDU). There are many ways that these 3 parameters can be set. In this study, we consider two cases which produce results in the two extremes: most-unsolicited and most-solicited data allowed by the iSCSI protocol. Most-unsolicited data implies that the *FirstBurstLength* is greater than or equal to the maximum write command request size that the initiator generates. Most-solicited data implies that the unsolicited data mode is disabled during the login negotiation, effectively equivalent to setting the *FirstBurstLength* to zero. In this mode, the target will notify the initiator when it is ready to receive data for a given command. This will give the target more control in the receive buffer allocation, but it will introduce extra round trip

delays as compared to the fully unsolicited mode.

3. Experimental Setup

In this paper, we study the effect of the aforementioned parameters on iSCSI performance for different network types between the initiators and the targets. Note however that we did not study scenarios with multiple TCP connections per iSCSI session because targets and initiators that support this feature are not yet widely available. In addition, the test configurations we study do not have multiple paths.

In order to isolate the effect of the network latency and not to be affected by the idiosyncrasies of different commercial products, we used a WAN/MAN emulator. This allowed us to vary the network latency while keeping all other parameters unchanged. More specifically, our experimental setup consisted of an open source software initiator by Cisco running on a 933 MHz two processor Intel Pentium III SMP machine, and an Intransa IP5000 iSCSI target, interconnected by a LANforge ICE WAN emulator by Candela Technologies. As traffic generators, we used two open source tools, *xdd* for block IO and *ettcp* for tcp traffic.

In our experiments, we set the network bandwidth to the OC3 rate, 155Mb/s. This rate is the maximum supported by the WAN emulator. Since this paper's focus is network performance, we configured the IP5000 in write-back mode and we set the traffic patterns such that all reads are served from the cache. This allowed us to eliminate any possible disk IO bottleneck.

We studied four values of round trip latency: 0 ms as baseline, 2 ms for MAN and 10 and 50 ms for WAN. In addition, we performed some LAN measurements by using a Gigabit Ethernet connection between the initiator and target, without the LANforge.

4. Results

4.1 Effect of the TCP Window Size

We first studied the effects of TCP window size setting. By using the default 64KB settings of the Linux kernel 2.4.19, we obtained the results shown in Table 1. The first row corresponds to data being sent from the iSCSI initiator machine to the target, and the second row corresponds to the data sent in the other direction. As can be seen, even at the small values of round trip latency, the default TCP window size settings are inadequate to maintain a good level of throughput. We then increased the maximum send and receive window sizes to 10 MBytes for both the initiator and the target, and achieved the wire speed for all the latency values under consideration as shown in Table 2. In the remainder of this paper, we kept the maximum window sizes at 10 MBytes.

Table 1: TCP throughput results in MBytes/s with default TCP send and receive window sizes (64KB)

Transfer Direction	Round Trip Latency			
	0 ms	2 ms	10 ms	50 ms
I → T	19.0	18.7	4.4	0.9
T → I	19.0	13.5	3.2	0.7

Table 2: TCP Throughput results in MBytes/s with maximum TCP send and receive window sizes set to 10 MBytes

Transfer Direction	Round Trip Latency			
	0 ms	2 ms	10 ms	50 ms
I → T	19.3	19.3	19.3	19.3
T → I	19.3	19.3	19.3	19.3

4.2 Effect of the iSCSI Parameters

After eliminating the TCP bottleneck, we studied the effect of the iSCSI parameters. Table 3 presents the iSCSI throughput results for most solicited writes using different iSCSI command window and request sizes. It is clear that the throughput is adversely affected when the product of window and request size is small.

Table 4 shows similar results to Table 3, but for most unsolicited writes. Clearly, the elimination of the extra round trip delays help to improve the throughput.

Table 3: iSCSI throughput results in MBytes/s for most solicited writes

Request Size	Window Size	Round Trip Latency			
		0ms	2ms	10ms	50ms
1KB	1	1.3	0.2	0.05	0.01
	32	11.3	3.3	0.8	0.2
8KB	1	7.7	1.7	0.4	0.08
	32	19.0	18.9	5.8	1.2
64KB	1	19.2	10.9	2.9	0.6
	32	19.2	19.3	19.3	4.5
256KB	1	19.2	19.2	7.8	1.7
	32	19.3	19.3	19.3	4.7

Table 4: iSCSI throughput results in MBytes/s for most unsolicited writes

Request Size	Window Size	Round Trip Latency			
		0ms	2ms	10ms	50ms
1KB	1	2.2	0.4	0.09	0.02
	32	17.4	6.6	1.5	0.3
8KB	1	10.3	3.2	0.7	0.2
	32	19.2	19.2	11.6	2.5
64KB	1	19.3	17.6	5.4	1.2
	32	19.3	19.3	19.3	4.8
256KB	1	19.3	19.3	11.6	2.5
	32	19.3	19.3	19.3	4.9

Table 5: iSCSI throughput results in MBytes/s for reads

Request Size	Window Size	Round Trip Latency			
		0ms	2ms	10ms	50ms
1KB	1	2.3	0.4	0.1	0.02
	32	17.6	8.0	2.3	0.5
8KB	1	10.2	3.1	0.8	0.2
	32	19.2	19.2	19.1	4.7
64KB	1	19.2	17.2	5.4	1.2
	32	19.2	19.3	19.3	19.2
256KB	1	19.2	19.2	18.3	4.7
	32	19.3	19.3	19.3	19.3

Table 5 shows similar results for read requests. Considering that both the read requests and the unsolicited write requests involve one round trip latency per request, the results in Table 5 match the results in Table 4 in many cases. However, for some other cases, read throughput seems to significantly exceed the most unsolicited write throughput.

Finally, our LAN measurement results are shown in Table 6 for writes and reads, using various iSCSI command window and request sizes. It is interesting to note that even in a low latency LAN environment, the product of the iSCSI window size and request size impacts the performance significantly. In addition, the unsolicited writes provide a significant increase in performance.

Table 6: iSCSI throughput results in MBytes/s in Gb/s LAN environment

Request Size	Window Size	Most Solicited Writes	Most Unsol. Writes	Reads
1KB	1	3.4	5.5	5.4
	32	17.1	23.4	19.9
8KB	1	17.4	25.0	22.5
	32	68.6	84.3	72.3
64KB	1	56.1	65.9	61.3
	32	96.5	99.8	91.9
256KB	1	71.3	79.9	74.7
	32	97.3	100.4	98.4

5. Conclusions

We have observed that the default TCP parameter values are inadequate for the high speed MAN and WAN environments, and therefore require tuning. We have seen that the product of the iSCSI command window and request sizes has a very significant effect on the performance as well. Furthermore, using the most solicited writes has a major performance penalty, and should be avoided whenever possible. With appropriate performance tuning, the iSCSI and TCP protocols are capable of achieving good throughput in all types of networks.

6. Acknowledgements

We would like to thank Mr. Ben Greear for kindly allowing us to use the LANforge LAN/MAN/WAN emulator for this study. We would also like to thank Mr. Robert Gilligan and Mr. Kenny Speer for their valuable feedback.

References

[1] IETF Internet Draft, "iSCSI", draft-ietf-ips-iscsi-20.txt, J Satran et al, Jan 2003

[2] "A Performance Analysis of the iSCSI Protocol," S Aiken et al, *proceedings of 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies,* San Diego, CA, USA, Apr 2003, pp123-134

[3] "IP SAN – From iSCSI to IP-Addressable Ethernet Disks," P Wang et al, *proceedings of 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies,* San Diego, CA, USA, Apr 2003, pp189-193

[4] IEEE 802.3-2002 "Information Technology - Telecommunication & Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications," 2002, ISBN 0-7381-3089-3

[5] IETF RFC 2001, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," W Stevens, Jan 1997

H-RAIN: AN ARCHITECTURE FOR FUTURE-PROOFING DIGITAL ARCHIVES

Andres Rodriguez
Dr. Jack Orenstein
Archivas, Inc.
Waltham, MA 02451
Tel: +1-781-890-8353
e-mail: arodriguez@archivas.com, jorenstein@archivas.com

1 Introduction

Traditionally, systems for large-scale data storage have been based on removable media such as tape and, more recently, optical disk (CD, DVD). While the need for increased storage capacity has never been greater, the inadequacies of traditional approaches have never been more apparent. This is especially true for fixed-content data: new government regulations and increasingly competitive market pressures have converged to underscore the importance of finding long-term storage solutions for fixed-content data that offer ready and secure access, easily scale, and are relatively inexpensive.

1.1 Shortcomings of removable media archives

Archives that rely exclusively on removable media share the following shortcomings:

- An archive system that commits physical data to removable media is also captive to the specific hardware system that enables read/write access.. As technology changes, these systems inevitably tend towards obsolescence. It is questionable whether the devices that are used today to read tape or disk will still be available and viable years hence—never mind the availability and viability of the vendor itself!
- As the archive grows, access becomes increasingly cumbersome and time-consuming. Data is not always readily available when it is wanted. Moreover, the administrative overhead that is required to provide timely access is unacceptably high, and—for many organizations—prohibitively expensive.
- Government regulations reflect a rising demand to maintain large amounts of data over long periods of time, and to guarantee their authenticity. Removable data is especially vulnerable to physical mishandling and corruption, both through physical deterioration and outside intervention, whether inadvertent or deliberate.

1.2 An alternative model

In general, when digital data is bound to tape or disk, it ceases to be a digital asset; instead, it becomes simply a physical widget that contains bits, with all the drawbacks previously cited. Long-term mass storage of fixed-content data requires a new type of storage model, where the data's physical location is completely separate from its logical representation. In order to achieve this objective, digital data must be stored in a digital archive that is scalable, reliable, and highly available.

Today's best hope of realizing this model rests in a network—or cluster—of inexpensive servers such as IA-compatible machines that can run a full Linux distribution. This model offers the following advantages:

- Various protection schemes (RAID-5, RAID N+K) safeguard files from multiple, simultaneous points of failure in the network, and guarantee that their data remains continuously available.
- Within the network, the archive system autonomously enforces policies that are associated with the stored files. These policies include retention period, file protection, and content authentication.
- Gateways for standard protocols (HTTP, NFS, SAMBA, CIFS) provide over-the-wire access to the archive.
- The archive is easily extended: as new nodes enter the cluster, the archive automatically invokes its own load-balancing and protection policies, and redistributes existing storage into the new space accordingly.
- A network-based archive can facilitate updates to files so they stay current with the latest applications—for example, format changes that are required by new end-user applications. Data migration of this type, on the scale required for large archives, is virtually impossible to achieve in tape-based systems.
- The data's actual location on the network is transparent to the user. During its lifetime in an archive, a stored file might be relocated across many network machines—or nodes—as the result of hardware upgrades, replacements, or load balancing. The reference to the file, however, remains constant, enabling users ready access to its contents without requiring knowledge of its physical location within the cluster.

1.3 Two architectures for online archives: RAIN and H-RAIN

In the last several years, various vendors have come forward with archive systems that implement the network approach just described. These all embody various implementations of RAIN (redundant array of independent nodes) architecture. RAIN archives are based on one or more clusters of networked server nodes. As nodes enter or leave the cluster, the cluster automatically adjusts by redistributing and, when necessary, replicating the stored files.

Currently, RAIN archives are typically delivered as proprietary hardware appliances, closed systems that are built from identical components. Evolution of these systems is carefully controlled by the vendor.

The architecture of H-RAIN—heterogeneous redundant arrays of independent nodes—differs from the RAIN architecture from which it evolved by making minimal assumptions about the archive's underlying hardware and software. In practice, this means that H-RAIN architecture can be implemented with commodity hardware. This relatively open architecture has two advantages over its RAIN progenitor:

- It adapts more readily to technological advances and site-specific contingencies. Administrators are free to replace components with superior hardware as it becomes available, thus improving storage capacity, performance, and reliability. Furthermore, they can choose among hardware options that best suit their requirements, such as CPU, memory, and disk capacity. For example, a cluster might be extended by adding new nodes with higher-performance CPUs, which can be used for CPU-intensive filtering operations. Incremental hardware additions and improvements might thereby measurably improve overall archive performance.

- Archive administrators can start small and scale up capacity incrementally simply by adding nodes as they are needed. Moreover, they are free to seek the best prices for storage cluster components. Given that component costs tend to decrease over time, cost-conscious administrators can reduce their average cost per gigabyte by spreading out purchases.

In general, an H-RAIN architecture enables users to upgrade their technical infrastructure while transparently migrating archive content to more up-to-date nodes. Improvements can be made incrementally, leaving the initial installation intact. As hardware prices fall, archive performance can be enhanced with better-performing nodes, and at lower cost.

2 Implementing H-RAIN architecture

Archivas' archive management system, Reference Information System (RIS), is based on the H-RAIN model. With RIS, organizations can create large-scale permanent storage for fixed content information such as satellite images, diagnostic images, check images, voice recording, video, and documents, with minimal administrative overhead.

In RIS' H-RAIN implementation, two features are salient:

- Distributed processing

- Autonomous management

2.1 Distributed processing

All nodes in a cluster are peers, each capable of running any or all of services that an archive requires. A cluster can be configured so archive services are distributed in a way that best serves the enterprise's storage requirements.

For example, a cluster can configured symmetrically—that is to say, each node runs the same processes and daemons, including a portal server, metadata manager, policy manager, request manager, and storage manager. Each node bears equal responsibilities for processing requests, storing data, and sustaining the archive's overall health. No single node becomes a bottleneck: all nodes are equally capable of handling requests such as put and get operations. Furthermore, in the event of node failure, any other node can take over responsibility for the data that was managed by the failed node, so that user access to this data remains unaffected.

Alternatively, a cluster might be configured so that various services are distributed asymmetrically across different nodes. For example, if read requests are especially heavy

for a given archive, several nodes might be dedicated solely to request management and run multiple request managers and metadata managers, in order to maximize throughput to other nodes that store the physical data.

2.2 Autonomous management

Through policies that are associated with archived files individually, and the cluster collectively, the archive can manage itself without human intervention. Policies are set for the archive on initial configuration, and can (optionally) be set for individual files as they are archived. Taken together, these policies determine the archive's day-to-day operation. Through a policy manager that executes on each node, the archive monitors its own compliance with current policies, and when lapses occur, takes the appropriate corrective action.

For example, in the event of a failed disk or node, the system determines what data is missing and how best to restore it from data on the remaining healthy nodes, so that the protection policy for these files is fully enforced. Similarly, the system prohibits removal of an archived file before its retention period has elapsed.

Human intervention is rarely warranted, and usually only in response to system warnings that require outside action—for example, notification the cluster load has crossed a specified threshold, requiring the addition of new nodes.

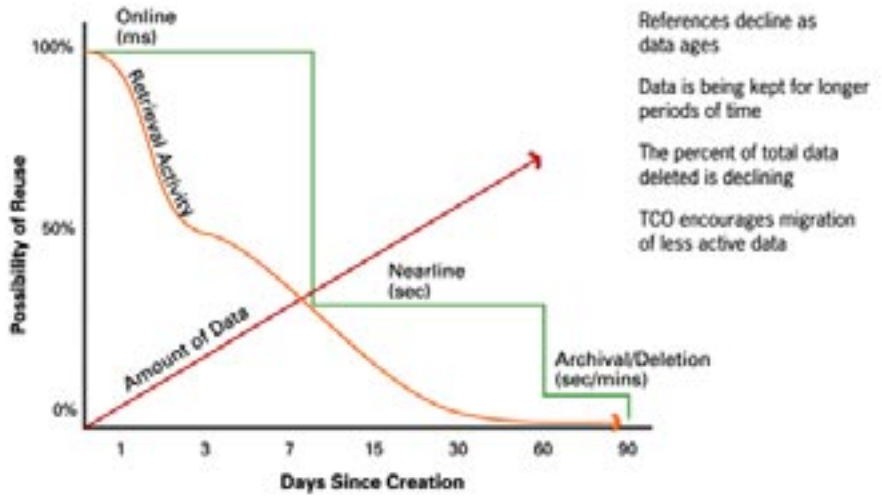
Four attributes characterize archive self-management:

- **Self-configuring:** Setting up large archive systems is error prone. An archive comprises networks, operating systems, storage management systems, and, in the case of RIS, databases and web servers; getting all these components to run together requires teams of experts with a myriad of skills. An autonomous system simplifies installation and integration by setting system configuration through high-level policies.
- **Self-protecting:** Policies that enforce document retention, content authentication, and file protection combine to protect an archive from loss of valuable digital assets.
- **Self-healing:** Serious problems with large-scale archives can sometimes take weeks to diagnose and fix manually. When the faulty device is finally identified, administrators must be able to remove and replace it without interrupting ongoing service. Autonomous systems can automatically detect software and hardware malfunctions in a node, and safely detach it from the archive. Further, because data is replicated across many nodes in the cluster, the failure of one or more nodes has no impact on data availability. Archivas' distributed metadata manager can find an alternative source for any data that resides on a failed node.
- **Self-optimizing**: Storage systems, databases, web servers and operating systems all have a wide range of tunable parameters that enable administrator to optimize performance. An autonomous system can automatically perform functions such as load balancing as it monitors its own operation.

3 Extending the H-RAIN model

With its H-RAIN architecture, RIS is capable of integrating with storage systems that use removable media such as tape or optical disk. In this scenario, the tape system is seen by RIS as simply another set of storage nodes; the physical location of data is managed by an RIS storage manager implementation that is specifically targeted to tape-based storage.

This capability is critical for a multi-tier storage and migration strategy, where data is stored in whatever medium best serves external access requirements. For example, a file that is frequently accessed should be archived in primary storage on a high-performance disk, while data that is rarely used can be stored on relatively low-performance media such as tape. Further, it is likely that access requirements for a given file will not remain constant, especially if the file is retained for a long period of time. In general, references to most types of data significantly decline as the data itself ages, as shown in the following figure:[1]



With the advent of government regulations such as the Sarbanes-Oxley Act, enterprises are required to archive increasing amounts of reference data, and retain them for ever longer periods of time. In order to keep storage costs down, it is increasingly important that archive systems respond to changing access requirements by moving data easily from more expensive disk-based media to less-expensive removable media. By encompassing both disk-based and tape-based storage and providing a unified interface to both, RIS can provide a smooth migration path for aging data. Furthermore, RIS policy mangers can automatically manage the migration, as determined by archive-wide or file-specific policies.

4 Conclusion

A digital archive system that is based on H-RAIN architecture offers the most economical, scalable, and effective solution for large-scale storage of reference data. Policy-based management minimizes administrative overhead while it provides the most reliable way to

achieve an archive's most important requirements: availability, reliability, and content authentication.

By extending the H-RAIN model to encompass any network node, including those that interface to tape-based systems, the potential exists to implement a multi-tier system where data is stored on the medium that best suits access requirements, and can easily be migrated to another medium as those requirements change.

---

[1] Fred Moore, "Information Lifecycle Management," Horison Information Strategies (http://www.horison.com)

A NEW APPROACH TO DISK-BASED MASS STORAGE SYSTEMS

Aloke Guha
COPAN Systems, Inc.
2605 Trade Center Drive, Suite D
Longmont, CO 80503-4605
Tel: +1-303-827-2520, Fax: +1-303-827-2504
e-mail: aloke.guha@copansys.com

Abstract

We present a new approach to create large-scale cost-effective mass storage systems using high-capacity disks. The architecture is optimized across multiple dimensions to support streaming and large-block data access. Because I/O requests in these applications are to a small fraction of data, we power-cycle drives as needed to create a very high-density storage system. Data protection is provided through a new variant of RAID, termed power-managed RAID™, which generates parity without requiring all drives in the RAID set to be powered on. To increase bandwidth, the system employs several concurrency mechanisms including load balancing I/O streams across many parallel RAID controllers. A number of optimizations are made to the interconnection architecture and caching schemes that lowers the cost of storage to that of typical automated tape library systems while exploiting the performance and reliability advantages of disk systems. Recent results at COPAN Systems have proven the viability of this new storage category and product.

1. Background

Growing reference information, regulatory data and rich media archives [1] are providing new impetus for mass storage systems. Unlike transaction applications, these systems access a small fraction of the data infrequently or on a scheduled basis. One such application is backup and restore that is characterized by large-block streaming writes and infrequent reads. With their need for large-scale storage, these applications are very sensitive to cost. Historically, they have been addressed by automated tape libraries. While concerns with performance, lack of data protection, and the reliability of tapes [2] have always favored disk systems, their limited capacity and high cost have prevented them from replacing tape. The advent of high-capacity SATA drives and their cost approaching that of tape motivates us to revisit this design issue.

Existing approaches to high-capacity storage have been incremental, typically substituting Fibre Channel (FC) disks with lower cost ATA disks in standard RAID arrays. This does not result in increased storage density or the cost per unit storage of tape libraries which usually have a 3X advantage over disk systems. Therefore, a fundamentally different approach is required.

We believe the best approach is to use an application and workload-driven architecture to provide the performance and reliability of disks at the scale and cost of tape.

2. A New Approach: Application Specific Storage System

The specific needs of archival storage applications help define the architectural constraints. These include:

- I/O requests to large block with sequential or predictable access

- Performance metrics based on data rates (Mbytes/sec) and not on I/Os per sec (IOPs)
- Time to first byte should be in ms to seconds, desired for mission-critical systems

Given the above, we can eschew many common but complex features of traditional disk storage architectures:

1. *No need for large primary RAM cache*. Since the storage is not used for high-transaction I/O, there is little need for large shared caches, except for reads.
2. *No need for high-speed switched interconnection from host to disks*. With more tolerance to latency than transactional systems, access from these applications do not benefit from non-blocking switched connectivity.
3. *No need to access all the data all the time*. With access to a small fraction of the data (e.g., <5% in tape systems), a majority of disks could be taken off-line if the mean latency is bounded. In addition, most writes can be done sequentially.
4. *No need to limit capacity based on interconnection bandwidth.* Since the capacity to data rate ratio is high, the ratio of total drive bandwidth to interconnect bandwidth can be higher than traditional disk systems.
5. *Ensure data availability*. Given the scale of data they maintain, archival storage must have high reliability, data protection and availability.

COPAN Systems' architecture is based on power management of a large number of SATA drives. The basic concept was first used in the MAID (massive array of idle disks) project [3] that examined tradeoffs in disk power consumption and performance. The results showed that a MAID with cache can effectively support most reads from a large database archive.

To meet enterprise class storage needs using a power-managed disk design, we had to satisfy multiple criteria. These included data protection, scalable capacity, high bandwidth, storage and data manageability, small footprint, and a cost equal to or better than tape storage costs. This introduces a number of design guidelines:

1. *Provide parity based redundancy*: Tradeoff redundancy with effective cost of storage. RAID 1 provides 100% redundancy, but it also doubles the cost per unit storage.
2. *Ensure data protection and performance with drives power-cycled.* Optimize tradeoffs between power cycling, storage density, performance, redundancy and cost. Keeping 30%-50% of drives online implies 10X more data online than tape systems.
3. *Ensure that I/O rate and parity protection are maintained, even when disks are in transition during the power-cycling of individual drives*.
4. *Provide ways to scale total bandwidth of the storage system*

An implicit design objective was the optimal packing and configuration of the drives. The architecture that works best for a given volume turns out to be a three-level interconnect. If the total number of drives in the system is N, then N is decomposed into a 3-tuple, such that

$$N = s.t.d$$

where s is the number of storage enclosures or shelves, t is the number of "stick", or column of disks, in the each shelf unit, and d is the number of disk drives in each stick in a shelf (Figure 1). All I/O requests to SCSI disk volumes arrive over FC links at the system controller which maps the logical volume to physical volumes on shelves connected by FC.

If physical constraints of the rack can be satisfied, *N* can be chosen to support very large storage capacities in a single system. The packaging of drives must also provide adequate heat dissipation so that the disks operate at or below the specified ambient temperature.

We provide brief descriptions of the data protection, performance and reliability of the system.

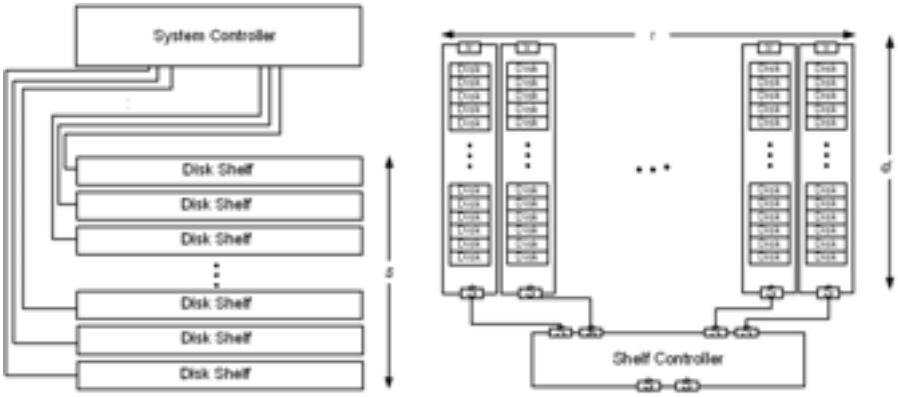


Figure 1. Three-tiered storage: decomposition into shelf, stick and disk

2.1. Efficient Data Protection: Power-Managed RAID™

Data protection using parity can be provided by power-cycling either full RAID sets or individual drives within the RAID set. COPAN Systems supports both approaches. However, power-cycling individual drives within the RAID set, termed power-managed RAID™ (PM-RAID™), has the advantage of keeping more RAID volumes on-line (assuming nominally 1 volume per RAID set) and reducing power swings.

In PM-RAID™, all data is written sequentially from drive to drive and the parity drive is fixed. At a minimum, only one data drive and the parity drive are powered on. When writing initially to drive D0, the data on the parity drive P will also be D0 if the parity drive was initialized to zeros. When the writes exceed the capacity of D0, the second drive D1 is powered up and data is written to it, while drive D0 is powered down. At this point, the parity drive P will contain the XOR of the earlier data written to D0 and the new data written to D1. Similarly, after the writes to the third drive, the parity drive will contain the XOR of the data from drives D0, D1 and D2. Thus, after writes to all data drives in the RAID set, the parity drive will indeed be the parity for all drives in the RAID set. On the failure of a data drive, all drives in the RAID set will be powered on to reconstruct the failed data drive as in a normal RAID reconstruct.

Figure 2 shows data written sequentially using a stripe size *s* of 1. If more bandwidth is desired, a larger stripe size *s*, $1 \leq s \leq n$ in an *n*+1 RAID, should be used. The most power-efficient and least bandwidth case is $s = 1$ with only 2 drives powered on, while the maximum bandwidth case is $s = n$ when all *n*+1 drives are powered on. Note that the $s = n$ case corresponds to traditional RAID 4 [4]. PM-RAID™ is therefore the most generalized version of RAID 4 under power constraints.

Various configurations of the number of on-line volumes and associated bandwidth are possible with PM-RAID™. For example, if there are 1000 data drives in the system that are set up as 4+1 RAID sets with a budget to power only 25% of all drives, the maximum number of volumes that can be powered is 125, or 50% of all volumes or 2X the drive fraction

powered on. Each volume can be accessed sequentially at a data rate possible from one disk drive.

PM-RAID™ utilizes a few always-on mirrored drives that maintain information on volume metadata, drive health, and read and write caches. Unlike previous approaches on disk on disk caching [5] used for performance, write caching is used to ensure that parity generation and I/O data rates can be sustained during power transitions of the drives. In general, including the spare and the metadata drives, less than 30% of all drives can be kept online while maintaining high bandwidth given sufficiently large number of disks in the system.

To ease storage management, the drive power management is transparent to the end user or application. The stripe width of the PM-RAID™ as well as any striping of the user's volumes across RAID sets in the shelves is managed by the system and not the user.

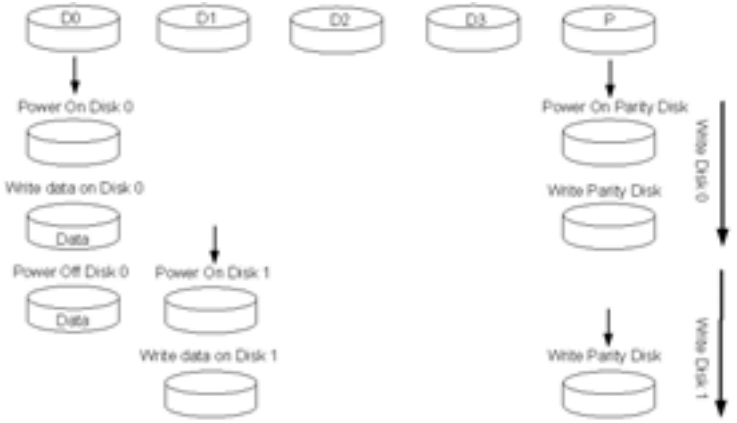


Figure 2. Writes in a power-managed RAID™

2.2. Increasing Performance: Concurrent I/O

Power management limits the number of drives powered on at any time. This limits the total I/O per storage shelf. To increase aggregate bandwidth, we use a number of concurrency techniques.

First, because the system controller has independent FC connections to storage shelves (Figure 1) and each storage shelf contains its own RAID controller, write and read bandwidth is increased *s*-fold with *s* shelves. Each shelf RAID controller can stripe data across up to *t* sticks.

Second, since each volume on the shelf is minimally comprised of 1 active data drive, the bandwidth can also be increased by striping (RAID-0) logical volumes from the system controller across volumes in the shelves.

Third, more I/O concurrency between the shelf controller and the SATA drives is provided in the stick using a custom SATA data and command router. This router provides concurrent access and command queuing of I/O to all *d* drives in a stick. This enables high-bandwidth I/O operations across a large number of drives in the shelf.

Finally, we use disk caches to cost-effectively ameliorate latency effects of disk spin-up, if there is no a priori information on access patterns. When write requests can be scheduled, as in a backup, a currently powered-down disk can be scheduled to power up with sufficient lead

time before the end of file is reached on the current drive. When the write requests cannot be scheduled, the data can be redirected to a spinning drive cache and later staged to the true disk target. Read disk caches also store data from all data drives. A read request to a powered-off drive is directed to the read cache, while the target drive is powered up. In the case of a read cache miss, our current measurements indicate that the read penalty is well below 10 seconds.

We also note that for streaming applications that access large blocks of data, the read latency of a few seconds on a cache miss is usually dwarfed by data transfer time.

2.3. Increasing Data Reliability: Effect of Power Management

Using high-capacity ATA drives and high-density drive configuration to create PB-sized storage systems raises the importance of data protection and data availability. We address the issue in many ways.

First, power-cycling the drives has a direct impact of increasing disk life and therefore system reliability. With 1000 drives and a drive MTBF of 400,000 hrs, the expected first failure of a drive is 18 days. Such low MTTF implies frequent drive swap-outs that is not be acceptable for most data centers. With PM-RAID™, if drives are powered with an average duty cycle of 25%, the net effect is to extend the effective drive MTTF to 4X the nominal value, or 1.6M hours, greater than typical SCSI or Fibre Channel.

Second, we closely monitor the total power cycles, also known as contact start-stops (CSSs) since all drives have a specified maximum CSS. We therefore ensure during operations that all drives are all well below the CSS threshold specified by the drive vendor.

Third, to accommodate drive failures that can put a large amount of data at risk, we use a unique proactive replacement mechanism. By continuously monitoring drive attributes, embedded controller intelligence determines whether a drive susceptible to failure should be replaced. Proactive drive replacement enables recovering data from the drive before failures, allowing the system to protect data even as failing drives are replaced.

3. Results

At the time of writing, we have demonstrated end-to-end I/O from the user to the drives. We have proved two important concepts. First, PM-RAID™ and the interconnect architecture perform as expected, with linear scaling of bandwidth with increasing I/O load. With the current design, 1000 MB/s or more bandwidth is possible in a single system. Second, the implementation shows that close-packing of drives in the system kept environmental attributes within specifications so data reliability is assured to be better than traditional disk systems. More detailed results will be presented at the conference.

REFERENCES

[1] Fred Moore, Are You Ready for MAID Technology? July 8, 2003, Computer Technology News, http://www.wwpi.com/lead_stories/070803_3.asp

[2] Bob Cramer, 'It's the restore, stupid!" April 23, ComputerWorld, http://www.computerworld.com/hardwaretopics/storage/story/0,10801,78483,00.html

[3] Dennis Colarelli, Dirk Grunwald et al, The Case for Massive Arrays of Idle Disks (MAID), Usenix Conference on File and Storage Technologies, Jan. 2002, Monterey CA.

[4] David A. Patterson, G. Gibson, and Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," SIGMOD 88, p. 109-116, June 1988.

[5] Y. Hu and Q. Yang, DCD -- Disk Caching Disk: A New Approach for Boosting I/O Performance. Proc. of the 23rd ISCA95, Philadelphia, PA, 1995.

Multi-Tiered Storage - Consolidating the differing storage requirements of the enterprise into a single storage system

Louis Gray

BlueArc Corporation
Corporate Headquarters
225 Baypointe Parkway
San Jose, CA 95134
USA
info@bluearc.com
Voice 1-408-576-6600
FAX 1-408-576-6601
http://www.bluearc.com

Abstract

Performance has always come at a steep cost in the world of enterprise storage, with flexibility often taking a back seat when it comes to IT purchasing decisions. In fact, storage has typically been bought and deployed as a "one size fits all" solution, regardless of the kinds of applications the network is running. Such an approach has required multiple servers to address multiple needs, resulting in significantly higher costs and a much greater degree of management complexity.

BlueArc's Multi-Tiered Storage (MTS) solution changes all this. MTS offers storage performance and consolidation while supporting different types of storage within the same network-attached storage (NAS)-based system, according to the specific requirements of the applications. For the first time, a combination of high-performance online, moderate performance nearline and infrequently accessed archival data can be configured in a single, seamless NAS platform—a BlueArc SiliconServer.

This paper looks at the issues surrounding today's growing storage needs in enterprise and project applications, examines the need for a multi-tier storage solution, explains BlueArc's Multi-Tiered storage platform and demonstrates how the system is applied to storage applications.

Today's Issues

Online data is doubling each year, but disk prices are falling rapidly. Now storage access, infrastructure and software costs are the dominant factors in the rising cost of storage.

Today's business need for scalable storage broadly fits into two categories - enterprise and project storage. In both of these applications, storage needs are growing exponentially.

Enterprise Storage

Enterprise user home directories are increasing more than two-fold annually, as presentations and documents become more graphics rich, email traffic increases and as employees need to keep more data at their fingertips to help their company prosper in a highly competitive and demanding marketplace. For example, it was not too long ago that a 2 Megabyte PowerPoint presentation was considered large, while now we see large presentations approaching 10 Megabytes apiece, and the average size nearing 2 Megabytes. Factors such as this, combined with employees' needs to retain additional storage in their home directories and mail archives is driving the average enterprise IT manager to prepare for a four-fold increase in storage space simply to keep pace with the organization's needs.

Project Storage

Work on digitized feature films, particle physics projects, bioinformatics/genetic research, seismic research, digital design, broadcast, medical imaging, CAD and other projects already require very significant levels of storage. Business demands and new enabling technology (e.g. low-cost Linux cluster super-computing) are driving for faster project completion and the analysis of even more detail, driving the need for performance network access to multiple terabytes of storage. The recent adoption of low-cost ATA disk arrays in many data centers offers affordable online storage or caching of archive data. In many applications this brings strong productivity benefits. Immediate online availability of archived data enables it to be quickly searched and re-used, rather than abandoned on low-performance archive tapes. In a typical enterprise, this combination promotes growth beyond tens of terabytes of online storage.

Infrastructure costs are becoming dominant in scaled storage applications

Attempts to deal with this level of growth are stretching current storage architectures to the limit. While Fibre Channel storage arrays are still a significant cost factor in expanding storage needs, the simple cost of storage arrays themselves is now not the prime contributor to growth costs. The infrastructure costs surrounding storage expansion (backup strategies, server proliferation and administration costs) are poised to become highly dominant factors for growth.

24 by 7 access to data

24 by 7 access to storage is being threatened by extending backup windows. Elimination of backup windows is the much-needed solution.

A difficult issue that enterprise and project IT managers face is the business mandate for 24 by 7 access to stored information. In the face of expanding storage needs, IT managers would like to be in a position to extend backup windows to accommodate for increased growth rather than contract them, enabling administrators to scale each server further, rather than proliferating servers. But backup activity cannot spill into peak business hours

if acceptable service is to be maintained. Traditional server architectures cannot run backup copy activity and continue to serve files without a severe drop in performance, which only becomes more difficult as service levels remain high around the clock.

Additionally, fast restore times and disaster recovery are key – the cost of downtime to businesses is high and IT managers need to strike the right balance in this area of the IT strategy, devising what amount of history to keep online and where it should be stored.

The introduction of large low-cost ATA storage arrays has changed the landscape for data restore and disaster recovery. With large, low-cost storage as the target for backup data and prime source for immediate backup history, both backup and restore time are reduced when compared to traditional tape backup. While online disk storage does not replace the high integrity holding data copies off site, disk storage offered as "virtual tape" or replicated intermediate storage is highly beneficial for reducing backup times and restore times. Savings and benefits from low-cost storage must not be competing with a higher cost, more complex infrastructure to access it.

Changing the rules for Storage

Today's issues demand a different architectural approach to storage.

- The infrastructure costs associated with scaling storage must be reduced.

  Enterprise and project storage needs will continue to expand. Raw storage costs are reducing significantly, but infrastructure costs associated with increasing storage capacity remain high. Proliferation of servers, backup and restore times, backup costs, storage networking complexity, and the administration overhead associated with storage infrastructures need to be tackled and contained.

- Match storage to applications without imposing large infrastructure costs.

  The arrival of resilient arrays of low-cost and high capacity ATA disks heralds the introduction of staged backup, which reduces both backup and restore times and lets servers scale much further. Low cost of storage and very high capacity are key elements for this application, since even the lowest performing disks are significantly faster than tape access. ATA storage fits this need. However, the performance characteristics of this type of storage are not high enough for database and transaction intensive applications. In these cases, Fibre Channel disks deliver appropriate performance at a higher cost.

- 24 Hour availability.

  Set-aside backup windows are no longer acceptable in today's around the clock businesses. They must be removed or significantly reduced, requiring a storage platform that can be backed up without affecting user access performance, even in

peak activity periods. Replication options must also operate without affecting user activity. Restore capabilities need to be very fast and disaster recovery needs to be quick and simple. Access to storage needs to be highly available, but redundant path architectures deployed to achieve this must be uniform across all types of storage without increasing infrastructure costs.

- Scaling by server proliferation is too expensive.

  Performance limitations in traditional servers restrict the amount of storage supported by a single server. Physical scaling limits for direct or SAN-attached storage cannot ultimately be avoided, even by offloading backup from the server. NAS appliances and NAS gateways extend these limits and are becoming more important as installed levels of storage continue to expand. Traditional NAS is more cost effective but lacks full integration

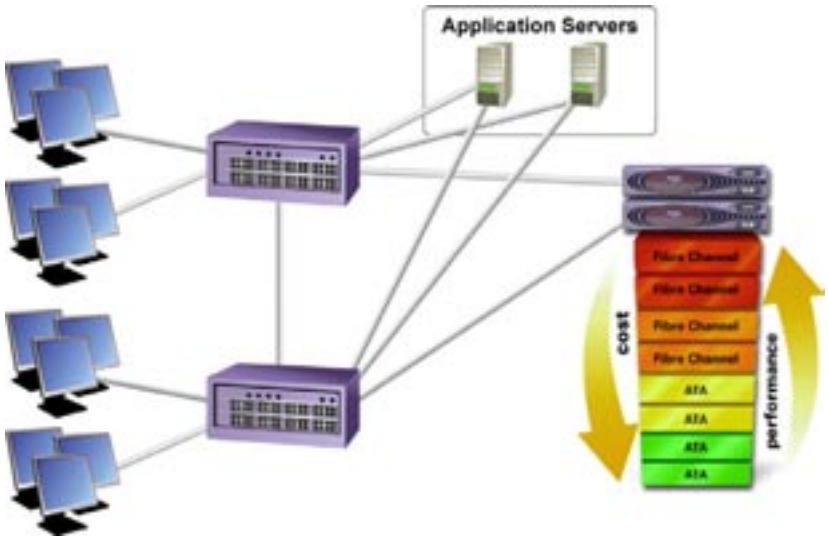
The role of NAS is changing. Switched Gigabit Ethernet as a network backbone expands the role of NAS in today's networks, while resilient network switch architectures support vast bandwidths with high availability. A Gigabit-switched backbone can support user traffic, application server disk access and archive traffic with ease. Furthermore, network traffic can be shaped and prioritized with standard tools in the Gigabit backbone switch, should the need arise.

With all this functionality included, the cost of commissioning a second and independent storage network needs careful thought. By removing file and archive storage from the SAN, server and SAN networking costs can be saved. For this reason, NAS has emerged as the optimal architecture for primary file storage and for online archive applications, which the availability of low-cost ATA storage has enabled. NAS also challenges the role of SAN as prime storage for database and other application servers.

Traditional NAS solutions have dedicated functionality. Performance NAS filers support primary file storage, and cost-competitive systems support general storage needs with the option of dual server heads offering high availability and fast fail-over in the event of a network or NAS component failure. Slower, cost-effective single head NAS systems are used for archive applications, built to support low-cost ATA storage generally in blocks of 2- 12 terabytes.

For a customer to require a combination of archive storage and primary storage is quite common, but to further reduce costs, an integrated approach, with all storage types supported in a single NAS system, has been sought but has not been found, as regrettably such an approach would demand unsustainable scalability and performance from the software-based architecture on which traditional NAS filers are based. The approach would also introduce performance issues related to backup, since is it hard to conceive that tape archive copying would be completed during today's shrinking backup windows. Decreased user access performance when running archive traffic during business hours adversely affects business productivity.

BlueArc - Scalable, high performance Multi-Tiered Storage with low Infrastructure costs



BlueArc Network Attached Storage is based on the SiliconServer, a NAS head with hardware data movement, similar to network backbone switches. BlueArc's architecture has the performance and scalability to support an integrated storage solution, with hardware data movement to maximize throughput.

BlueArc also solves the backup traffic problem, because of the high internal bandwidth in the server and because movement of data between network and storage is prioritized over disk to disk and disk to tape activity, should contention for resources arise.

BlueArc's Multi-Tiered Storage, with SiliconServer scalability and performance, consolidates storage to a single system. The system changes network storage rules by:

- Reducing the infrastructure costs of scaling
- Offering multiple tiers of varied cost and performance storage, without imposing large infrastructure costs
- Reducing the number of servers through consolidation and replacement
- Enterprise class continuous availability storage

The enhanced productivity derived from MTS is seen by customers in fields such as Broadcasting, Post Production, Manufacturing Genetic Research, Government Research, Mapping, University projects and many more.

The use of hardware to move data within the SiliconServer is extended to backup and disk-to-disk copies over Fibre Channel. The SiliconServer uses hardware data movement in the implementation of NMDP, the standards based backup protocol for NAS. Coupled with the ability to schedule a snapshot of the file system prior to making a backup, this capability has three distinct advantages:

1. The traditional problem associated with slowed server performance when running backup activity is removed and there is no need for a backup window.

    - Network data movement is prioritized over NDMP traffic
    - NDMP is supported by all backup software applications
    - Backups run faster because of speed of hardware data movement

2. Tape drive efficiency
    - Backups of a frozen on-disk snapshot can be spooled to tape as a background activity, if necessary running 24 hours per day to get maximum scalability out of tape drives. This delivers two to three times the scalability from each tape drive compared with running the drive only during a backup window. This mode of operation is fully supported by standard data backup software applications.

3. Backup copies can be stored online for immediate access

    - Fast restore and disaster recovery by referencing an online snapshot of the backup, rather than referencing the archive tape. SiliconServer file system snapshots store a complete image of the file system at the time the snapshot was taken. The snapshot retains a block level image of the file system. As files subsequently change, the snapshot stores only the blocks that have been modified since the snapshot was taken. Snapshots are highly efficient in terms of the amount of disk space they occupy, therefore it costs little to retain a week or more of daily backup copies online for fast restore should it be necessary. Backup copies on disk are not only highly desirable for business continuance in the event of a disaster or accidental deletion, but they also reduce the number of required tape library tape slots - if backup history is held online, there is no need to leave recent backup tapes online in the tape library.

Shared File Storage

BlueArc's Multi-Tiered storage is mounted as a network share/export. This means that multiple application servers can share the same data using standard Windows and UNIX locks. The SiliconServer also supports secure sharing between UNIX and Windows environments, with permissions mapping and lock integration between the two environments. Data sharing is vital for imaging, Web applications, design, engineering, and research applications where centralized data needs to be shared between a number of application servers and clients. BlueArc's Multi-Tiered Network Attached Storage is the optimal, strategic choice for these applications.

Managing Scalability in Object Storage Systems for HPC Linux Clusters

Brent Welch
Panasas, Inc
6520 Kaiser Drive
Fremont, CA 94555
Tel: 1-510-608-7770
e-mail: welch@panasas.com

Garth Gibson
Panasas, Inc
1501 Reedsdale Street, Suite 400
Pittsburgh, PA 15233
Tel: 1-412-323-6409
e-mail: garth@panasas.com

Abstract
This paper describes the performance and manageability of scalable storage systems based on Object Storage Devices (OSD). Object-based storage was invented to provide scalable performance as the storage cluster scales in size. For example, in our large file tests a 10-OSD system provided 325 MB/sec read bandwidth to 5 clients (from disk), and a 299-OSD system provided 10,334 MB/sec read bandwidth to 151 clients. This shows linear scaling of 30x speedup with 30x more client demand and 30x more storage resources. However, the system must not become more difficult to manage as it grows. Otherwise, the performance benefits can be quickly overshadowed by the administrative burden of managing the system. Instead, the storage cluster must feel like a single system image from the management perspective, even though it may be internally composed of 10's, 100's or thousands of object storage devices. For the HPC market, which is characterized as having unusually large clusters with usually small IT budgets, it is important that the storage system "just work" with relatively little administrative overhead.

1. Scale Out, not Scale Up

The high-performance computing (HPC) sector has often driven the development of new computing architectures, and has given impetus to the development of the Object Storage Architecture. The new architecture driving change today is the Linux cluster system, which is revolutionizing scientific, technical, and business computing. The invention of Beowulf clustering and the development of the Message Passing Interface (MPI) middleware allowed racks of commodity Intel PC-based systems running the Linux operating system to emulate most of the functionality of monolithic Symmetric Multi-Processing (SMP) systems. Since this can be done at less than 10% the cost of the highly-specialized, shared memory systems, the cost of scientific research dropped dramatically. Linux clusters are now the dominant computing architecture for scientific computing, and are quickly gaining traction in technical computing environments as well.

Unfortunately, storage architecture scalability in terms of performance, capacity, and manageability have not kept pace, causing systems administrators to perform tedious data movement and staging tasks on multiple standalone storage systems to get data into and out of the Linux clusters where scalable resources are available. There are two main problems that the storage systems for clusters must solve. First, they must provide shared access to ever larger amounts of data so that the applications are easier to write and storage is easier to balance with the scaling compute requirements. Second, the storage system must provide high levels of performance, in both I/O rates and data throughput, to meet the aggregated requirements of 100's, 1000's and in some cases up to 10,000's of nodes in the Linux cluster. Linux cluster administrators have attempted several approaches to meet the need for shared files and high performance, supporting multiple NFS servers or copying data to the local disks in the cluster. But to date there has not been an effective solution to match the power of the Linux compute cluster for the large data sets typically found in scientific and technical computing.

1.1 Methods of Data Sharing in Clusters

Sharing files across the Linux cluster substantially decreases the burden on both the scientist writing the programs and the system administrator trying to optimize the performance of the system and control the complexity of cluster management. There are several approaches to providing shared data to a computing cluster:

• *Network file servers*. The NFS protocol and file server hardware impose a bottleneck between the clients that share the data and the disk resources that store it. As storage needs grow, additional file servers with their own disk resources must be added, creating storage "islands" and adding complexity for users and administrators alike.
• *Block storage* via SCSI on Fiber Channel (FC) provides good performance access for a small number of disks, but block storage is private to individual hosts making it generally unscalable. Systems like GFS [1] and GPFS [2], sometimes called SAN filesystems, can provide shared FC storage for compute nodes. However, the costs of FC adapters per node, FC switching with enough ports for all cluster nodes, as well as FC administrators, can significantly increase the cost of each node in the compute cluster. Moreover, SAN filesystems are also fundamentally block-based, and the overhead of managing block-level metadata in terms of lock messaging and distribution of block-level metadata limits scalability.
• *Peer-to-peer* systems share hard drives attached to individual compute clusters. However, the complexity and cost of providing storage redundancy prohibits permanent storage of valuable data in per node disks. Moreover, storing data in per node disks introduces complexity in load balancing because some nodes have much worse I/O performance depending on where the file was written.
• A new alternative is *Object-based Storage Architectures*. The Panasas object-based storage system is built from commodity parts, including SATA drives, standard processors and memory, and Gigabit Ethernet to provide a storage system with excellent price/performance characteristics. Its high performance file system can be shared among multiple compute clusters equally and with little overhead. It can also share the same

files with legacy engineering workstations using standard NFS and CIFS protocols. The entire storage system is available through one global shared namespace in a manner reminiscent of the Andrew File System (AFS) [3].

2. Object-based Storage Architecture

The Object-based Storage Architecture utilizes data Objects, which encapsulate variable-length user data and attributes of that data [4,5,6]. The device that stores, retrieves and interprets these objects is an Object Storage Device (OSD) [7]. The combination of data and attributes allows an OSD to make decisions regarding data layout or quality of service on a per-object basis, improving flexibility and manageability. The object interface is mid-way between the read-write interface of a block device, and the high-level interface of an NFS server. The core object operations are create, delete, read, write, get attributes, and set attributes. By moving low-level storage functions into the storage device itself and accessing the device through a higher-level object interface, the Object Storage Device enables:

- Dedicated resources to block-level management,
- Intelligent space management in the storage layer allowing the OSD to make late binding decisions on the allocation of data to the storage media,
- Data-aware pre-fetching, and caching,
- A natural paradigm for scaling the capacity and performance characteristics of the storage system.

OSD-based storage systems can be created with the following characteristics:

- Robust, shared access by many clients,
- Scalable performance via an offloaded data path,
- Strong fine-grained end-to-end security.

These capabilities are highly desirable across a wide range of typical IT storage applications. They are particularly valuable for scientific and technical applications that are increasingly based on Linux cluster computing which generates high levels of concurrent I/O demand for secure, shared files.

2.1 Implementing Files using Objects

An Object-based storage system includes metadata managers that coordinate data access from multiple clients and implement POSIX filesystem semantics over the object storage. They also provide location information and implement security policies. For scalable performance, the metadata servers are “out-of-band” of the data path between clients and storage nodes. The metadata servers retain strict control of what data clients can access because the OSDs provide secure access rights enforcement with every operation [8]. Studies indicate that even in demanding small file with random access workloads, these metadata servers handle less than 10% of the work associated with file access, with the rest going to the object storage devices [9], and that load balancing metadata service can be managed by partitioning the objects [10].

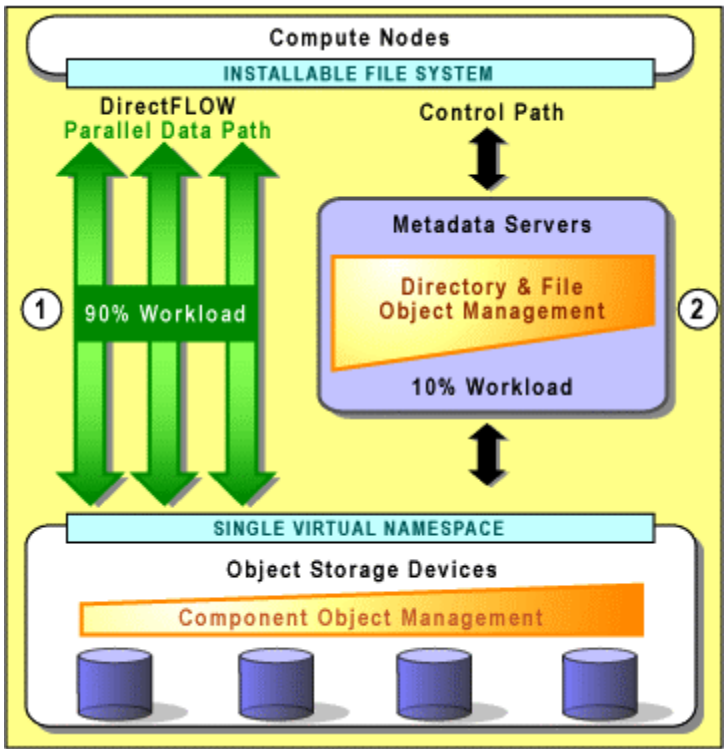


Figure 1. Object Storage Architecture

In the Panasas implementation, metadata managers are executed on server blades called DirectorBlades, and the OSDs are executed on server blades called StorageBlades. All permanent filesystem data storage is on the StorageBlades. The storage cluster can have a variable number of Director and StorageBlades depending on the workload requirements; workloads with legacy NFS and CIFS access need more DirectorBlades, while high-bandwidth, large file workloads need very few DirectorBlades.

The object interface has a core security protocol that allows safe concurrent access by multiple clients. Clients read directory objects to process filesystem pathnames. Directories map from names to object identifiers. Clients contact metadata servers to obtain location information (*maps*) and security capabilities (*caps*) that enable direct I/O access. These "maps and caps" are cached by clients so repeated access to files do not require interaction with the metadata manager. The OSDs verify security capabilities on every client access, so they enforce the policies implemented on the managers without having to know the details of POSIX or Windows ACL semantics. Our metadata managers also implement a lease-based cache consistency protocol so clients can efficiently cache file attributes and data, which provides further optimizations to file system access.

Individual files are striped across multiple StorageBlades (OSD). Striping files across objects allows high bandwidth access as well as protection from failures by using standard RAID techniques. The combination of file distribution across storage devices and multiple client access leads to very high aggregate performance to the shared storage system.

2.3 NAS Compatibility

It is also important that a new architecture such as Object-based storage be able to support NFS and CIFS compatibility for data sharing with desktops and other non-cluster computing platforms. The Panasas storage cluster does this in its DirectorBlades. DirectorBlades export standard NFS and CIFS interfaces, hiding the Object Storage Cluster from these legacy systems. Of course, a single NAS interface is a performance

bottleneck, but by deploying multiple NAS "filer head" interfaces in the storage cluster, even legacy applications see scalable performance, albeit less efficiently than through the Panasas file system protocol available on the Linux cluster nodes. Every DirectorBlade is capable of serving any file in any StorageBlade to any client. Panasas provides a global filesystem namespace, so clients only need a single mount point, which they can mount from any available DirectorBlade.

By providing a shared filesystem between the Linux computing cluster and the rest of the computing platforms, data management is dramatically simplified. For example, non-cluster hosts with tape drives can import data via multiple NFS access points in parallel, then the Linux computing cluster can access the data in place, and finally, non-cluster desktop applications can visualize and analyze the results from the computing cluster. In contrast, other approaches require explicit distribution of data to each node in the computing cluster, or management of multiple separate NAS systems.

3. Performance of Object-based Filesystems

There are four main components of the Panasas storage system: the client, the DirectorBlade, the StorageBlade, and the Shelf [11]. The client is a loadable kernel module that runs in the Linux compute nodes and implements a POSIX filesystem. It plugs into the VFS interface inside Linux. The StorageBlades have 2 SATA drives, a 1.2 GHz Pentium III processor, 512 MB memory, and 1 GE network port. The DirectorBlades have a 2.4 GHz Pentium 4 CPU, 4 GB memory, and 1 GE network port. Each shelf holds up to 11 Storage or DirectorBlades, and it has an integrated GE switch that provides up to 4 trunked GE ports out of the shelf. (A pass-through card provides 11 independent ports, but the numbers shown here use the integrated switch.) Each shelf also includes dual power supplies and a battery module, which together provide an integrated UPS function for the DirectorBlades and StorageBlades.

The bandwidth tests described below were run with 9 or 10 StorageBlades (OSD) per shelf, and in the bandwidth tests the DirectorBlades were mostly idle because most of their resources are reserved for NFS and CIFS. The goal of presenting these numbers is to give a general flavor of the scalability of the system performance. Results vary depending on the speed of the clients, the size of their I/O requests, and network topology. Typical clients in our tests have a single 2.4 GHz Pentium CPU and 1 GE network interface. Typical I/O requests in our bandwidth tests are 64 KB, and the network in our lab connects the systems and cluster under test through a high-end non-blocking Extreme Network BlackDiamond GE switch. In the bandwidth tests, files are always large enough that data must be streamed on and off the disk platters as opposed to being satisfied by cached data.

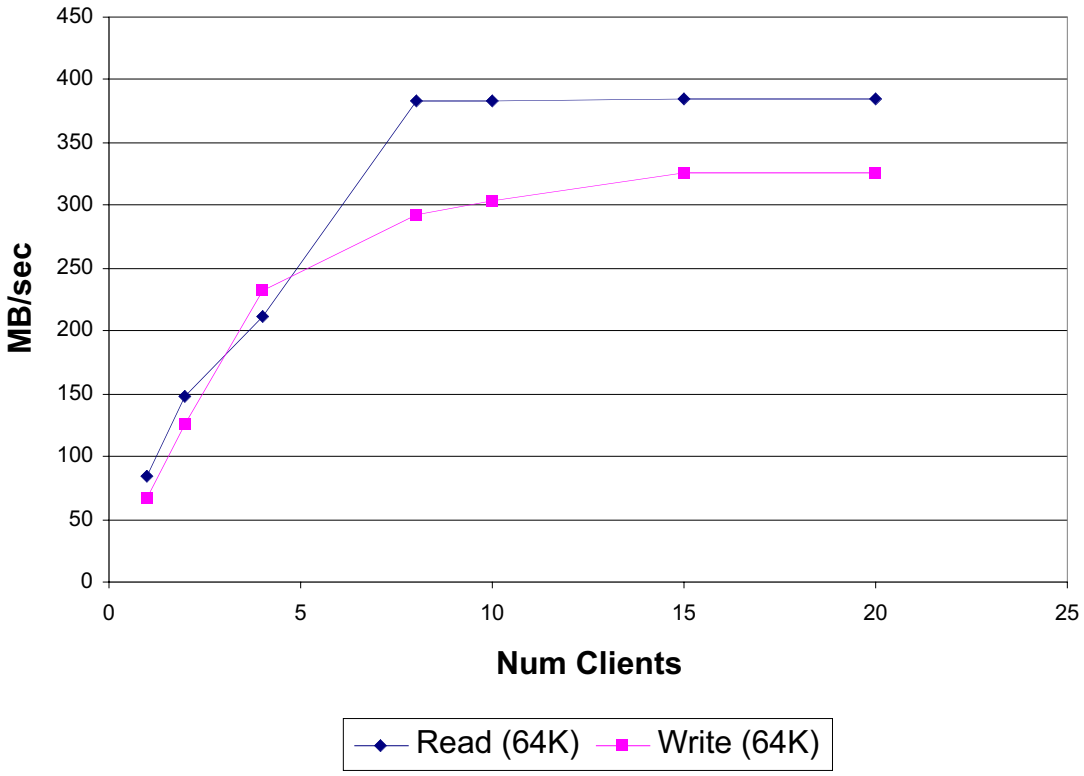


Figure 2. Bandwidth of 10 OSDs vs. the number of Clients

Figure 2 shows that aggregate bandwidth of one file per client in the same directory on a single shelf of 10 OSDs scales quite well as the number of clients increases until the shelf is providing about 380 MB/s. A single client can read from a file that is striped across 10 OSDs at about 90 MB/sec using a single GE port. As the number of clients scales up, but the storage resources remain the same, the aggregate bandwidth increases but the per-client bandwidth drops off. The less than linear scaling after about 8 clients per shelf is due to increasing load and contention at the storage devices that have to manage multiple I/O streams in parallel. There is also contention and a bottleneck in the shelf network as the data passes through the four trunked GE links providing a maximum of 4 Gb/s to each 10-OSD shelf. At saturation, each SATA drive is delivering a sustained bandwidth of just over 19 MB/sec split among the tens of clients pounding on it.

Write bandwidth follows a similar curve, with a single client writing at about 77 MB/sec and 10 clients able to write at 335 MB/sec. The write bandwidth peaks at less than the read bandwidth because the RAID engine runs at the client and so parity data flows over the network between the client and the storage nodes. For example, there is about 12% more data being written in an 8+1 RAID configuration than is reported in the bandwidth number. The advantage of moving RAID to the client is that it allows shared access to a scalable number of drives without the bottleneck of a traditional RAID controller [12]. In addition, the XOR computations RAID requires can be done efficiently using specialized

MMX instructions on the Pentium CPU, and their speed increases as client CPUs and memory systems get faster.

Figure 2 might led one to believe each shelf was a standalone 300-400 MB/s storage system. Not so. When multiple shelves are bound together by Panasas object storage software, total performance scales at 300-400 MB/s per shelf. Scaling when adding more shelves works quite well even if files are still only striped over 10 StorageBlades because the set of OSDs used to store each file are drawn from all OSDs on all shelves. The contention at any single storage device remains about the same.

Figure 3 shows several multi-shelf high bandwidth test results. For example, the test with 299 OSDs achieved 10334 MB/sec read bandwidth. There were 32 shelves and 151 clients, or about 5 clients per shelf. The per-shelf bandwidth of 322 MB/sec in this configuration is consistent with tests done with 5 clients against one shelf. (The large test used a larger application read blocksize than in the chart shown in Figure 2, so the numbers are not directly comparable.) As files are added, their data is spread across different subsets of OSDs automatically, on a per-file basis. This allows large numbers of clients to share many OSDs efficiently. Figure 3 shows that bandwidth scales quite well with the number of OSDs. In fact, because our tests are generally done with only about 5 clients per each 10-OSD shelf, Figure 2 indicates that the total bandwidth achievable from these systems can be higher. For example, the difference between the points at 116 and 118 OSDs is that the first test used 61 clients to achieve 3.1 GB/sec, while the second used 79 clients to achieve 3.9 GB/sec. That is about 50 MB/sec per client in both configurations. Finally, it is important to note that these data points were taken over time in our labs with varying configurations of clients, network topology, and software tuning, so each point is not strictly comparable. However, the overall result is that the system scales well, at about 15-20 MB/s per disk, when the number of clients is at least 25% as many as there are disks.

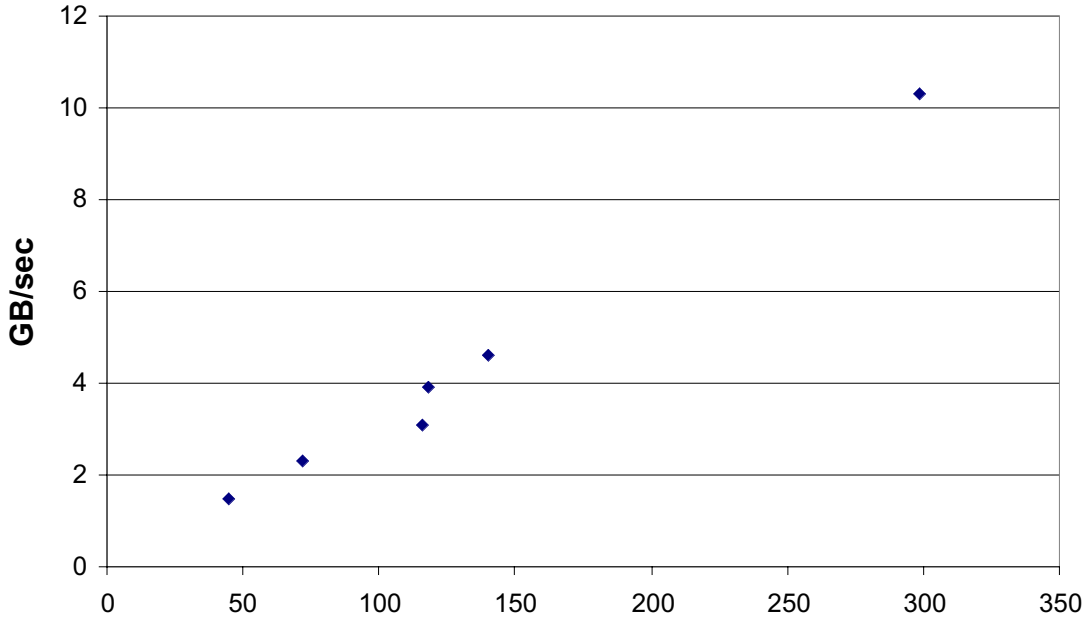


Figure 3. Scaling Aggregate Bandwidth with Number of OSDs

When files are striped very widely, contention increases because each OSD has connections to more clients. In a concurrent write test with 151 clients writing to a single, shared file striped across 198 OSDs, the write bandwidth was 2775 MB/sec. The OSDs were organized into 22 shelves with 9 OSDs each, for a per-shelf bandwidth of about 126 MB/sec.

For legacy, non-cluster computers, access is through NFS or CIFS. Because Panasas DirectorBlades offer multiple NFS servers as a single storage pool, the underlying scalability of Object Storage is made available as a scalable NAS system. Figure 4 shows two results from the industry standard SPEC SFS benchmark [13][1]. First we ran the SFS test against a 5-shelf system with 10 metadata managers and 45 OSDs, providing a single rack 90-disk system similar to the high-end of dedicated monolithic NAS filers. This delivered an excellent throughput of 50,907 ops/sec at an Overall Response Time (ORT) of 1.67 msec. The average response time as a function of load is shown in Figure 4 in the lightly shaded curve peaking at 50,000 ops/sec. ORT is a weighted average of the average response time at each of 10 load points. To show how NFS scales, we also show the results from a 30-shelf system with 60 metadata managers and 270 OSDs, which achieved 305,805 ops/sec at an ORT of 1.76 msec. While 300,000 ops/sec is much larger than any other reported SFS benchmark run to date, our real point is that a Panasas storage cluster with 6x the resources delivers 6x the workload at about the same response time profile.

[1] NFS ops/sec as measured by the SPEC benchmark. SPEC and the benchmark name SPECsfs97_R1 are registered trademarks of the Standard Performance Evaluation Corporation.

The SFS test has a uniform access requirement so each client is contacting every file server during the benchmark run. In this case each DirectorBlade is running two orthogonal functions: one is the metadata management for the object storage filesystem, the second is a client of that filesystem that is being exported via an NFS server module. The NFS server on each DirectorBlade accesses the metadata function on other directors as well as storage on all the OSDs.

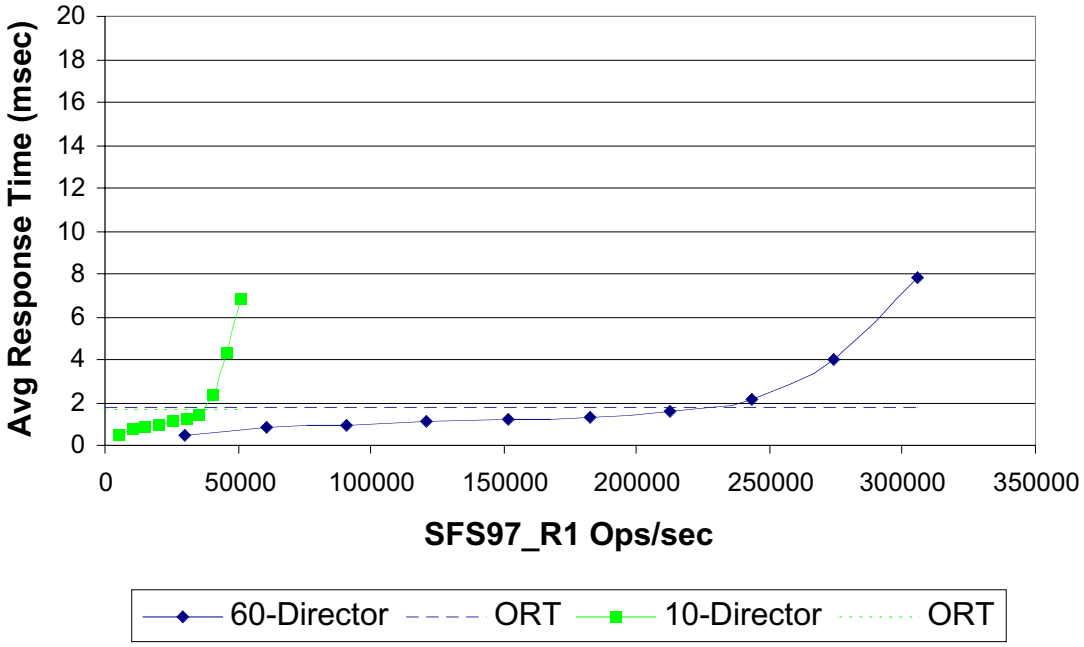


Figure 4. Results for 10- and 60-DirectorBlade Systems

4. Managing Large Scale Systems

The ideal scalable storage system is a large, seamless storage pool that grows incrementally without performance degradation and is shared uniformly by all clients of the system under a common access control scheme. As the system scales in size, however, issues arise in two general areas: traditional storage management and internal resource management. Both of these areas are affected by the distributed system implementation of the storage system itself. To external clients, the storage system should feel like one large, high-performance system with essentially no physical boundaries imposed by the implementation. Internally, the system must manage a large and growing collection of computing and storage resources and shield the administrator from chore of administering individual resources.

4.1 Traditional Storage Management

Traditional storage management issues include system configuration, monitoring system performance and capacity utilization, responding to failures, and adding and configuring hardware resources as the system grows. A scalable storage system should minimize the

burden these traditional storage management issues place on an administrator so the system can grow to very large capacities with undue operating costs.

The Panasas object-storage architecture simplifies capacity and device management by hiding the details normally associated with block devices such as LUN definition and Fiber Channel zone definitions. Instead, the filesystem is built from a collection of object storage devices (OSD) and metadata managers (Directors). The filesystem automatically stripes files across storage devices using RAID techniques to tolerate the failure of storage devices or individual objects (e.g., due to media errors). As more storage devices are added, files are striped more widely. As stripes become wider, additional parity objects are introduced to limit the size of failure domains. A unique aspect of object-based storage is that RAID configurations are configurable on a per-file basis. For example, mirroring is more efficient for small write accesses, but has high capacity overhead compared with RAID 4 or RAID 5. Ordinarily RAID parameters are managed automatically by the system. For example, the Panasas filesystem uses mirroring for directories and small files, while larger files use RAID 5. However, in simplifying management we do not want to go so far that we hurt our users' ability to optimize performance. For this reason an optional a programming interface exposes stripe width and RAID parameters so MPI IO middleware layers can create files with specific bandwidth and reliability attributes to best match application requirements.

New capacity is added simply by adding one or more new StorageBlades to the system. However, this can create a capacity imbalance among the StorageBlades, new blades less full than older blades, so the system actively rebalances capacity across StorageBlades. This is done efficiently by transparently moving component objects. For example, if files are striped across N StorageBlades, then each file is composed of N component objects, one component on each StorageBlade. When a new StorageBlade is added to the system, the system selects component objects from each of the existing StorageBlades, and moves those component objects onto the newly available StorageBlade. This reduces the capacity utilization across existing StorageBlades and fills up the new StorageBlade. The active balancer runs in the background at a low priority. The filesystem blocks the balancer from using files that are being used, and if an application happens to access a file that is currently being rebalanced, it is temporarily blocked from using the file. The application's access proceeds automatically once the balancer has finished moving the component object.

Backup and restore is obviously important for any storage system. One nice benefit of a high performance, shared storage system is that multiple backups can proceed in parallel. This helps the administrator reduce the backup window to manageable levels, even with very large systems.

Monitoring and configuration is done via a central management console that has a Web interface as well as a command line interface. Any feature accessible via the GUI is also accessible via the CLI. It usually turns out that the GUI is best for new users, offering a simple display of performance monitoring information, and a quick overview of the system. However, for large systems and more experienced administrators, it can become

tedious to configure the system one click at a time. Instead, a CLI that can be scripted (we have a TCL-based shell) can be a real time saver and an enabler for site-specific monitoring and data collection.

4.2 Internal Resource Management

Because the storage system is itself a cluster of computers working together to provide service, there are internal management issues such as resource discovery, software upgrade, failure detection, power and thermal management. These internal issues should be handled automatically by the system, yet provide the administrator with monitoring, failure reporting, and robust failure handling.

The Panasas storage system is IP-based, and its system configuration includes a block of IP addresses that is managed by an internal DHCP service. This runs on an alternate port so it will not conflict with the customer's existing DHCP infrastructure. The Panasas DHCP protocol is extended with additional information about device serial numbers, types (Storage or Director), software revision level, and physical location (shelf and slot). As blades boot up the system discovers their type, location, and software version and automatically builds its configuration database. StorageBlades are automatically added to the storage pool as they come on-line, so provisioning a running system just requires physical addition of StorageBlades.

The external view of the system is through a single DNS name. Clients mount the file system from this single name, and their filesystem accesses are automatically directed to the appropriate DirectorBlade as they access different directories. Of course, I/O access goes directly between clients and StorageBlades using the maps they get from DirectBlades during access control checks. In high availability configurations, the system DNS name is mapped to a set of IP addresses, and clients can contact any of these addresses to mount the filesystem. For NFS and CIFS load balancing, Panasas provides a delegated DNS name server to distribute legacy clients across DirectorBlades.

Software versions are maintained uniformly across the storage cluster to avoid awkward compatibility issues. Each blade boots from its own drive to avoid massive congestion on a netboot server during system startup. Software upgrade is achieved with a two-phase installation operation where all blades install a new version and reach a commit point in the first phase. Only if all blades are ready does the system commit to the new version and restart. Filesystem clients pause for the duration of the commit and restart, which is about 5 minutes regardless of the size of the storage cluster. When new blades are added to a system they are checked for hardware compatibility, and they are automatically upgraded to the same version as the rest of the system.

Power and thermal management is important for any high-density cluster installation. The Panasas blades are housed in a shelf that has dual power supplies and a battery module that together provide an integrated UPS. The UPS protects the blades against power surges and brown outs. If AC power is lost completely, the blades are signaled and use battery power to write out cached data and safely shutdown the system. By providing

an integrated UPS and power management in every shelf, the system can be aggressive about caching data in main memory without burdening the administrator with building a foolproof data center-scale UPS and scaling it up as the system grows. The StorageBlades accumulate data and metadata updates and periodically flush these in a log-like fashion. This lets the system optimize disk arm seeks and provide very high write throughput even during shared workloads. Thermal monitoring is also integrated, and the system will proactively take itself offline if external temperatures rise and cause blades to overheat. The system is able to differentiate between power or thermal failures and disk or blade failures so it can respond appropriately.

5. Conclusion

The ability of storage systems built on the Object Storage Architecture to scale capacity and performance addresses a key requirement for HPC Linux clusters. Panasas' Object-based storage cluster demonstrates scalability with 32-shelf systems providing 30x the bandwidth of a single shelf, and 30 shelf NAS benchmarks providing 6x the throughput of 5-shelf runs of the same benchmark.

While we want performance and capacity to grow linearly as resources are added to a storage cluster, we do not want administrator effort to grow anywhere near linearly. Object Storage Architectures are designed to abstract physical limitations, making virtualization easier to provide, so that larger systems can be managed with little more effort than small systems. Panasas object-based storage clusters use distributed intelligence, a single namespace interface, file-level striping and RAID, and transparent rebalancing to realize the manageability advantages of Object-based Storage.

References

[1] Soltis, Steven R., Ruwart, Thomas M., et al. *The Global File System*, proc. of the Fifth NASA Goddard Conference on Mass Storage Systems, IEEE, 1996.

[2] Schmuck, Frank, and Haskin, Roger. *GPFS: A Shared-Disk File System for Large Computing Clusters.* Proc First USENIX conf. on File and Storage Technologies (FAST02), Montery, CA Jan 2002.

[3] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., West, M.J, *Scale and Performance in a Distributed File System*, ACM Transactions on Computer Systems, Feb. 1988, Vol. 6, No. 1, pp. 51-81.

[4] Gibson, G. A., et. al., *A Cost-Effective, High-Bandwidth Storage Architecture*, 8th ASPLOS, 1998.

[5] Azagury, A., Dreizin, V., Factor, M., Henis, E., Naor, D., Rinetzky, N., Satran, J., Tavory, A., Yerushalmi, L., *Towards an Object Store*, IBM Storage Systems Technology Workshop, November 2002.

[6] *Lustre: A Scalable, High Performance File System*, Cluster File System, Inc. 2003. http://www.lustre.org/docs.html

[7] Draft OSD Standard, T10 Committee, Storage Networking Industry Association (SNIA), ftp://ftp.t10.org/t10/drafts/osd/osd-r05.pdf

[8] Gobioff, Howard. *Security for a High Performance Commodity Storage Subsystem*. Carnegie Mellon PhD. Dissertation, CMU-CS-99-160, July 1999.

[9] Gibson et. al. *File Server Scaling with Network-Attached Secure Disks*, ACM SIGMETRICS, June 1997, pp 272-284

[10] Brandt, S., Xue, L., Miller, E., Long D., *Efficient Metadata Management in Large Distributed File Systems*, Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies, April 2003.

[11] http://www.panasas.com

[12] Amiri, K., G.A. Gibson, R. Golding, *Highly Concurrent Shared Storage*, Int. Conf. On Distributed Computing Systems (ICDCS2000), April 2000.

[13] http://www.spec.org/sfs97r1/

The Evolution of a Distributed Storage System

Norman Margolus
Permabit, Inc.
One Kendall Square, Bldg. 200
Cambridge, MA 02139
nhm@permabit.com
tel +1-617-995-9331
fax +1-617-252-9977

Abstract

Permabit is a software company that makes a storage clustering product called Permeon. Permeon grew out of the need to reconcile a vision of the future of globally distributed, secure, private and robust storage with near-term marketplace realities. This paper discusses the evolution of the ideas embodied in Permeon.

1 Introduction

When Permabit was founded in June of 2000, it was directed towards making the "disk in the sky" Storage Service Provider (SSP) idea practical. Storage clients would send permanent data off into the network, with the guarantee that this data would be kept safe and private. Most of our initial concerns were interface issues between the storage client and the storage system, and were largely independent of the structure and implementation of the actual storage. It was only later that we focused on distributed storage clustering software.

2 Content Addressed Storage (CAS)

To make remote network storage immediately practical for ordinary users, we had to first address the issue of the availability and cost of bandwidth. We observed that if a cryptographic hash of a block of data is used as the name for the block, then users of a widely shared Internet storage system should be able to backup much of the data on their PC's to a "disk in the sky" without sending much data. Not only could they avoid transmitting data that the storage system has seen before, but we could also avoid storing it separately for each user. This could make it practical for a large number of users to store large amounts of data remotely, enabling data sharing and remote access to their data.

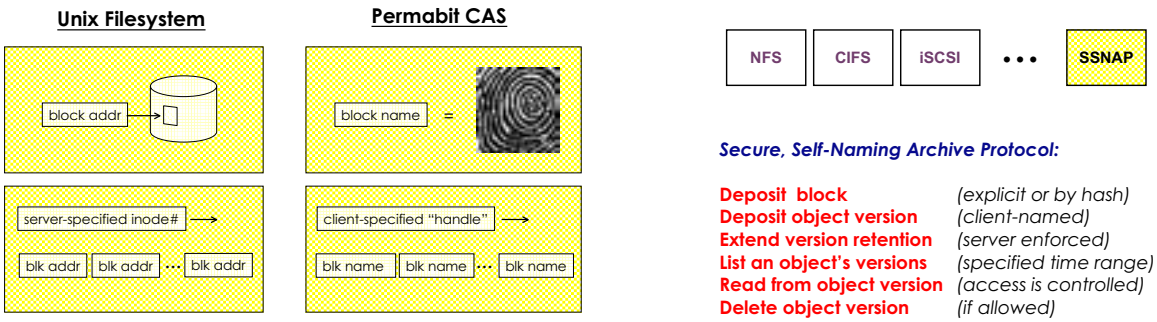


Figure 1: Content addressed storage (CAS) can save valuable bandwidth on the WAN, reduce storage requirements, and guard data integrity. In Permabit CAS, a cryptographic hash *fingerprint* of each data block is used as the address of that block, enabling storage sharing. An unshared metadata-level, analogous to the *inode* level in Unix, controls access and security. A secure network protocol (SSNAP) maintains privacy and thwarts IP-piracy.

2.1 Self-Encryption

CAS bandwidth and storage savings depend on sharing storage between unrelated users. This seems to require that the storage providers have complete access to all of the data. This is something that is unlikely to be palatable to the end user for privacy reasons, and is also a significant potential source of liability to the storage provider.

This issue can be dealt with using self-encryption: each block of data stored can first be encrypted using a key derived deterministically from the unencrypted block (again using a cryptographic hash). Unrelated users will produce the same encrypted block, but no one without the source block can determine the key. As long as keys are never stored "in the clear" in the storage system, no one who has not had access to an unencrypted copy of the block can determine what it contains. The Farsite project[1] uses a similar idea, but only to save storage, not network bandwidth of the depositor.

2.2 Access Control

If knowing the hash of a block is a sufficient credential for being granted access to the block, the storage system has no real access control. For example, in a widely shared "disk in the sky" storage system, the hash corresponding to the contents of a newly released DVD might become widely broadcast. The storage provider needs a way to obey a court-order to remove illegitimate access to this content without removing legitimate access.

It is natural, for this reason, to introduce a metadata level which is analogous to the *inode* level in a Unix file system, as is illustrated in Figure 1. A small amount of unshared per-client and per-object metadata allows conventional access control and provides a place to keep privately-encrypted key information. Only the block-level is globally shared and self-encrypted. Blocks can only be read by reference to unshared per-client metadata.

2.3 Network Protocol

The SSNAP network protocol for communicating with the "disk in the sky" is summarized in Figure 1. Hash-named blocks of data can be deposited by name to save bandwidth, but they can only be read as part of an object which belongs to a particular client (to permit read-access control). Depositing blocks by hash-name may involve a per-client challenge, to ensure that the client actually has the block and not just a hash that someone has broadcast to them.

Objects are analogous to inodes in a Unix file system, except that the *object handle* (inode number) is supplied by the client. Both block names and handles are 32 bytes long. Objects can have multiple historical versions, allowing retention policies for protecting history to be enforced by the storage system. This is used to protect against accidental or malicious corruption. Server enforced record retention is also useful for automating record retention requirements mandated by government regulations and business best practices.

2.4 Security and Privacy

No attempt is made to anonymize user access. Privacy maintenance is based on not storing records that allow a stored object to be linked to a user when the object is not actually being accessed. Depending upon accounting requirements, whatever information is desired can be logged or not logged at access time. By not recording who owns what data, storage service providers can avoid having these (non-existent) records subpoenaed.

The globally unique name of an object is obtained by combining a namespace identifier with the handle constructed by the client—which the client ensures is unique within the namespace and unguessable. The handle and namespace are combined by the storage system using a one-way hash function to locate the object. To prevent objects that are not being accessed from being linked to users, handles are not retained by the storage system.

2.5 Embedding File Systems

SSNAP provides an object level interface at the equivalent of the Unix inode level. Client libraries build upon this, to support file sharing protocols with directories. A generalized *Portable Hierarchical File System* (PHFS) data format is defined, with an associated API, which allows file system metadata for different file systems to be embedded. Multiple historical versions of files are supported, to allow convenient file system snapshotting for backup purposes. File system snapshots can be copied into Permabit storage and presented through familiar file sharing protocols. Permabit storage can also be used directly for live file storage, with copy-on-write snapshotting providing backup. By breaking files up into hash-named blocks at natural boundaries (e.g., email attachment boundaries), the likelihood of storage sharing (block coalescence) is enhanced.

2.6 Enterprise Storage

Although SSNAP and PHFS were originally conceived with end-users in mind, most of the considerations that went into their design are at least as important for enterprises. Divisions within an enterprise would like to be able to share storage without compromising their

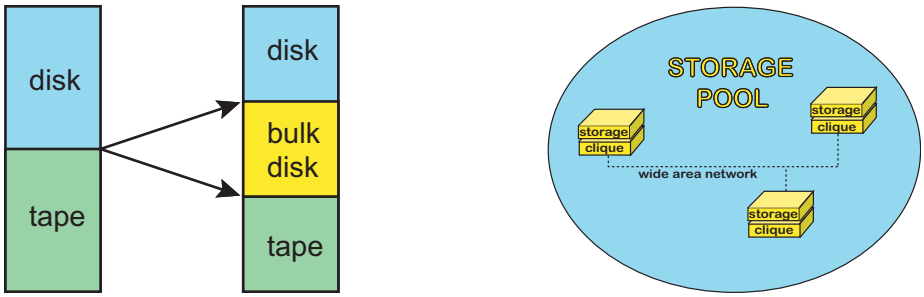


Figure 2: Distributed storage systems can displace tape as safe and economical long-term/large-scale storage. Geographically distributed storage cliques can protect redundant records from loss or corruption, and independently enforce record retention policies.

privacy or security. Storage-system enforced data retention is important in this context not only for preserving backup snapshots of a file system, but also for meeting government regulatory mandates for the long-term retention of email and other sensitive records.

The elimination of common storage is particularly interesting for enterprises. The Venti project at Lucent Bell Labs[2] showed that, with the combination of avoiding duplication of common blocks of data and compressing the blocks, all of the daily backup data for each of two large shared file systems *accumulated over a decade* could be stored on disk using only slighly more storage than the current active data on the file systems.

3 Distributed Storage

The considerations discussed above apply regardless of the nature of the "disk in the sky." Since hash-named data is a natural match for scalable clustered storage, and stimulated by the low cost of ATA disks and commodity PC's, we decided to develop our own storage clustering software.

3.1 Bulk Storage

Storage systems are naturally hierarchical. For example, RAM memory systems use slower but more economical memory for bulk storage, while caching the most active data to hide most of the latency. With the raw media cost per gigabyte of ATA disks below that of tape, it seems natural to expect that the bulk of storage will soon be low-cost disk storage, used within a storage hierarchy (Figure 2). But low cost does not mean just cheap disks. Most of the cost of disk storage today is in its management: adding and removing storage, keeping it working, backing it up and preparing for disaster recovery. Bulk disk storage needs to be self-managing, self-healing, self-backing and disaster tolerant in order to qualify as low cost storage.

To be disaster tolerant, the bulk disk storage must be distributed: information destroyed at one location must be redundantly represented elsewhere. In considering costs, however, it is important to also keep bandwidth in mind. The cost of bandwidth on the WAN is several orders of magnitude greater than the cost of bandwidth on a local network. This makes a two level system attractive, in which local clustering of economical standard hardware

constitutes the first level. In the Permeon system the local cluster provides fast recovery after failure of a server, fast access for nearby clients, high aggregate disk and network bandwidth, and cooperative caching behavior.

3.2 Scalability

Overlay network routing schemes developed for academic distributed storage systems depend on statistical allocation of storage capacity. This does not work well for small clusters. Permeon *cliques* are designed to scale up indefinitely starting from a very small number of servers, allowing a low-cost entry point. For this reason, a table-based routing scheme is used within cliques, in which address ranges for hash-named data (and replicas) are assigned and reassigned as servers are added or removed. Data replicas are created and moved automatically as address assignments change.

Because of high WAN bandwidth costs, if a clique is destroyed there is a large benefit in being able to recreate it using data resident at one or a small number of locations. This leads to a multi-clique scheme in which the data from each clique is redundantly represented at some small set of other cliques.

3.3 Portals

To provide convenient access to the distributed storage system, Permeon provides portal servers which present the clique storage using familiar file sharing protocols. These portals can live behind firewalls, providing secure access to shared storage cliques. A web-based storage management console runs on the portals.

4 Conclusions

The distributed "disk in the sky" storage systems of the future will be the descendents of decentralized bulk-disk storage of the post-magnetic-tape era which make storage inexpensive by solving the costly storage management, repair, data corruption and disaster tolerance problems. In particular, these storage systems will have to enforce data-retention policies, and do so independently in different locations, in order to provide useful backup and dependable archiving. Systems which save bandwidth and storage space by avoiding transmitting or storing duplicate data have a distinct advantage in the near term. Privacy and security provisions are already important today, and will be vital to making the systems suitable for widely shared access.

References

[1] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," ICDCS, July 2002.

[2] Sean Quinlan and Sean Dorward, "Venti: a new approach to archival storage," *First USENIX conference on File and Storage Technologies*, 2002.

List of Authors